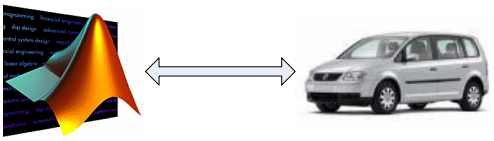
Institut für Regelungstechnik

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

Prof. Dr.-Ing. W. Schumacher
Prof. Dr.-Ing. M. Maurer
Prof. em. Dr.-Ing. W. Leonhard

Sicherstellung des deterministischen Verhaltens bei der Kopplung mehrerer Softwaretools innerhalb einer Gesamtfahrzeugsimulation



Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von	:	Dipl.-Ing. Lamjed Bouabdallah
aus (Geburtsort)	:	Tunis
eingereicht am	:	16. Oktober 2008
mündliche Prüfung am	:	10. Februar 2009
1. Referent	:	Prof. Dr.-Ing. Walter Schumacher
2. Referent	:	Prof. Dr.-Ing. habil. Dipl.-Math. Bernd Meinerzhagen

2009

Vorwort

Die maßgeblichen Richtlinien, die zur Entstehung dieser Dissertation führten, ohne den freien Raum der Kreativität einzuschränken, stammen zum größten Teil von Herrn Prof. W. Schumacher. Ich spreche ihm daher meinen herzlichen Dank aus.

Besonders dankbar bin ich für die tatkräftige Unterstützung von Herrn Goldau und seinem Team bei der Volkswagen AG. Die fachlichen Diskussionen dort bildeten die Grundlage dieser Arbeit.

Unterstützend waren ferner die Gespräche, Arbeiten und Anregungen von Institutsmitgliedern und Studenten, insbesondere nenne ich hier Herrn Dr. Trabelsi, Herrn Dr. Chouika und alle Studenten, die bis spät in die Nacht mitgewirkt haben, um Teile dieser Arbeit zustande zu bringen.

Meiner Familie und meinen Freunden, die mich sowohl in den „guten“ als auch in den „schlechten“ Zeiten unterstützt haben, gilt mein spezielles Dankeschön.

Inhaltsverzeichnis

Symbolverzeichnis

q	Generalisierte Lagekoordinaten
u	Generalisierte Geschwindigkeitskoordinaten
$T = T(q)$	Abbildungmatrix
$M = M(q)$	Generalisierte Massenmatrix, symmetrisch positiv definit, enthält Trägheitseigenschaften aller Körper
$h = h(q, u, t)$	Vektor der generalisierten inneren und äußeren Kräfte mit einwertigen Kraftgesetzen sowie aller gyroskopischen Kräfte
$W = W(q)$	Matrix der Kräfterichtungen
$\aleph$	Menge der Nebenbedingungen in Form von Gleichungen und Ungleichungen
Δt, h_x	Schrittweite einer Simulation
ω	Winkelgeschwindigkeit
e_x	Messfehler der jeweiligen Größe x
ϵ	Drehwinkel
x_c	Kontinuierlichen Zustände in der GSE
x_d	Diskrete Zustände in der GSE
y	Ausgang der GSE
$zcSignal$	Zero Crossing Signal aus dem Simulink-Block
S_x	Sensor x
c	Federsteifigkeit
D	Federdämpfung
M_a	Von außen angreifendes Moment
M_{ri}	Reibmomente
f	Freiheitsgrad eines Systems
T	Kinetische Energie
V	Potentielle Energie
L	Lagrange-Funktion
M_B	Bremsmoment
F_R	Reibungskraft

r_R	Radradius
v_F	Fahrzeuggeschwindigkeit
$\mu(\lambda)$	Reibkoeffizient
λ	Schlupf
GSE	General State of Equation
UML	Unified Modeling Language
LOTOS	Language Of Temporal Ordering Specification
SA/RT	Structured Analysis/Real Time
SADT	Structured Analysis and Design Technique
GSPN	Generalized Stochastic Petri Net
BDK	Boolesches Differentialkalkül
MKS	Mehrkörpersimulation
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
GUI	Graphical User Interface
ODE	Gewöhnliche Differentialgleichungen
Solver	Computerprogramm zur digitalen Gleichungslösung
ZC	Nulldurchgang (zero-crossing)
ZCD	Erfassung von Nulldurchgängen (zero-crossing-detection)
DLL	Dynamic Link Library
CORBA	Common Object Request Broker Architecture
OMG	Object Management Group
IDL	Interface Definition Language
TCP/IP	Transmission Control Protocol/Internet Protocol
HTTP	Hypertext Transfer Protocol
STEP	STandard for the Exchange Product Model Data
DMU	Digital Mock-Up
PDM	Product Daten Management
Co-Simulation	Kopplung von zwei Simulationssoftware auf Integratorebene
Integrierte Simulation	Kopplung von zwei Simulationssoftware auf Modellebene
ZC-HaSh	Zero-Crossing-HandShake

Kurzfassung der Dissertation

Die zunehmende Komplexität der heutigen industriellen Produkte, insbesondere in der Automobilbranche, weist den Simulationen eine immer wichtigere Rolle zu. Aus diesem Grund wurden in den letzten Jahren neue Tools und Simulationskonzepte erforscht und entwickelt, die es ermöglichen schon während der Konstruktionsphase aussagekräftige Informationen über ein Produkt zu liefern. Somit können wertvolle digitale Informationen für weitere Bereiche gewonnen werden und gleichzeitig werden teure Hardware-Prototypen und kostenintensive Revisionen an den ersten Serien vermieden.

In der vorliegenden Dissertation wird das Prinzip der modell- bzw. werkzeuggekoppelten Simulation als Erweiterung der neuen Modellierungskonzepte beschrieben und analysiert. Ziel dieser Beschreibung und Analyse ist die Sicherstellung des deterministischen Verhaltens der Simulation in Abhängigkeit der Schrittweite, so dass eine Verzerrung der Simulationsergebnisse vermieden wird. Das deterministische Verhalten ist dann vorhanden, wenn die Simulation identische Ergebnisse zu identischen Zeitpunkten liefert, wenn die Simulationsmodelle identische Stimuli zu identischen Zeitpunkten wie in der Realität erhalten. Ferner dürfen die Ergebnisse nicht von einander abweichen, wenn Änderungen an den Simulationseinstellungen vorgenommen werden, wie beim Typ des Gleichungslösers.
Als Beispiel dienen mathematisch berechenbare dynamische Modelle und andere komplexere Fahrzeugmodelle. Die rechnergestützte Implementierung der dynamischen Modelle bzw. Regelungsmodelle wird mit den bekannten Werkzeugen ADAMS von der Firma MSC und MATLAB von Mathworks durchgeführt.

Beim Entwurf der Gesamtmodelle stand stets die Beschreibung und die aufgabenspezifische Verbesserung der Interface-Struktur zwischen den Softwarepaketen im Vordergrund. In den meisten Fällen zeigt die Interface-Struktur ein gutes Verhalten bei der Simulation linearer Systeme; jedoch weist sie Lücken im Umgang mit Diskontinuitäten auf. Die Implementierung eines *Eventhandlers* behebt das Problem in diesen kritischen Situationen und spielt somit eine wichtige Rolle bei der Standardisierung solcher Schnittstellen. Dieses Lösungskonzept leistet daher einen sinnvollen Beitrag zur Realisierung einer deterministischen Gesamtsystemsimulation anhand eines Softwareverbundes. Diese Dissertation soll somit den Entwick-

lern helfen, zuverlässige und robuste Simulationsergebnisse zu bekommen, die zwingend zur Verbesserung des Produktlebenszyklus führen.

1 Einleitung

1.1 Motivation

Im Produktentwicklungsprozess gewinnen in der letzten Zeit rechnergestützte Entwicklungs- und Fertigungsmethoden in der Automobilindustrie immer mehr an Bedeutung. Grund dafür sind unter anderem die rasant gestiegenen Rechenleistungen der Computer, die neuen, verbesserten elektronischen und mikroelektronischen Komponenten und Systeme und die steigende Komplexität der Fahrzeuge als Gesamtsysteme. An die Automobilhersteller werden höhere und zum Teil völlig neue Ansprüche gestellt. Hierzu zählt in erster Linie die günstige Realisierung eines hohen Funktionsumfanges. Dazu kommen die neuen Vorgaben des Gesetzgebers und die verschiedenen Kundenwünsche bezüglich Sicherheit, Zuverlässigkeit, Umweltschutz, Komfort und Kommunikation. Letztendlich bemühen sich die Konzerne um die technologische Führung im harten internationalen Wettbewerb, bei dem Zeit und Kosten sehr wichtige Aspekte sind.

Die Integration von neuen Simulationskonzepten im Entwicklungsprozess, wie anhand des V-Modells in Abbildung 1.1 erläutert, verkürzt den Weg zu einem marktreifen Produkt. Bislang wurden die Simulationen innerhalb einer Ingenieursdisziplin mit den vorhandenen Tools durchgeführt. Doch mit zunehmender Komplexität und wachsender Kommunikation zwischen den einzelnen Elementen ist der Bedarf an übergreifenden Simulationskonzepten unübersehbar. Die Aufgabe einer Gesamtsystemsimulation ist es hauptsächlich, schon während der Konstruktionsphase aussagekräftige Informationen über das Fahrzeug zu liefern. Hinzu kommt die Vermeidung von kostenintensiven Revisionen an der ersten Serie [14].

Erste Schritte im Sinne von integrierenden Simulationskonzepten, welche das Fahrzeug als ein digitales Produkt abbilden, wurden bereits gemacht. Die Realisierungen können fast immer nur mit Hilfe mehrerer Tools unternommen werden. Grund dafür ist die Bewahrung der Flexibilität, der Einfachheit und der Exaktheit bei der Erstellung der Modelle innerhalb eines jeweiligen Tools. Die einzelnen Komponentenmodelle werden in den Teildisziplinen wie z.B. Mechanik, Feinmechanik, Elektronik, FE-Analyse, Elektronik und Regelung, getrennt

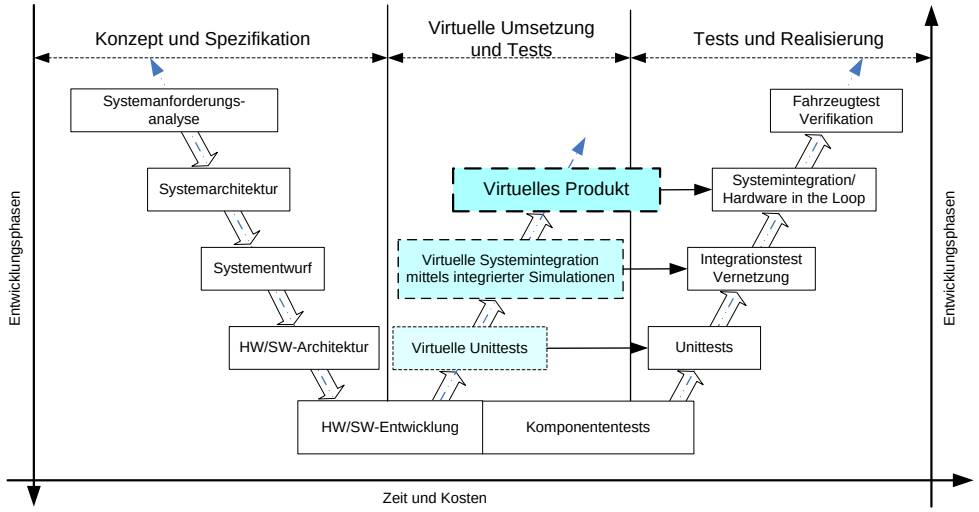


Bild 1.1: Integration der Simulationen in das V-Modell des Entwicklungsprozesses

entwickelt und in dem für die jeweilige Disziplin vorgesehenen Werkzeug simuliert. Daraus entstehen beim Zusammenfügen dieser Teilmodelle zu einer Gesamtsystemsimulation verschiedene Probleme, welche von der Art des Gesamtsimulationskonzeptes abhängen.

Diese Konzepte werden in zwei Kategorien unterteilt: Erstens werden Lösungen anhand der Kopplung der Simulationswerkzeuge realisiert, bei der zwei Gleichungslöser simultan aber selbständig die systembeschreibenden Gleichungen jeweils für die eigene Teildisziplin lösen. Die zweite Alternative ist die Kopplung der Werkzeuge auf Modellebene. Hier wird ein in einem Tool A erstelltes Modell in ein Tool B exportiert, in dem die Lösung aller Systemgleichungen anhand des eigenen Solvers berechnet werden.

Die Schwierigkeit bei der Realisierung der oben genannten Konzepte liegt vorwiegend in der Auslegung der Schnittstelle, welche den großen Anforderungen gewachsen sein soll.

Zu den Anforderungen zählt ein robuster Aufbau dieser Schnittstelle, so dass ein reproduzierbares Verhalten bei der Simulation entsteht. Doch spätestens bei der Umsetzung dieser Art von Schnittstellen stellt der Anwender fest, dass ein Verhalten bei der Simulation komplexer Modelle entsteht, das einem „deterministischen Chaos“ ähnelt. Im deterministischen Chaos ist das System zwar vollständig durch Gleichungen beschrieben, diese sind aber nicht linear und damit ist die Entwicklung unter realen Bedingungen nicht völlig vorhersehbar. In extremen Fällen ist eine sensitive Abhängigkeit von den Anfangsbedingungen bemerkbar. Somit verursachen kleine Änderungen der Startwerte sehr unterschiedliche Endergebnisse. Das Phänomen ist als *Schmetterlingseffekt* in der Wissenschaft bekannt [2], [31]. Die folgende Abbildung 1.2 aus [31] zeigt ein demografisches Beispiel für solche Systeme Das Modell

ist durch:

$$x_{n+1} = r \cdot x_n \cdot (1 - x_n)$$

beschrieben. x_n ist dabei eine reale Zahl zwischen 0 und 1. Sie repräsentiert die relative Größe der Population im Jahr "n". Die Zahl x_0 steht also für die Startpopulation (im Jahr 0). "r" ist immer eine positive Zahl, sie gibt die kombinierte Auswirkung von Vermehrung und Verhungern wieder.

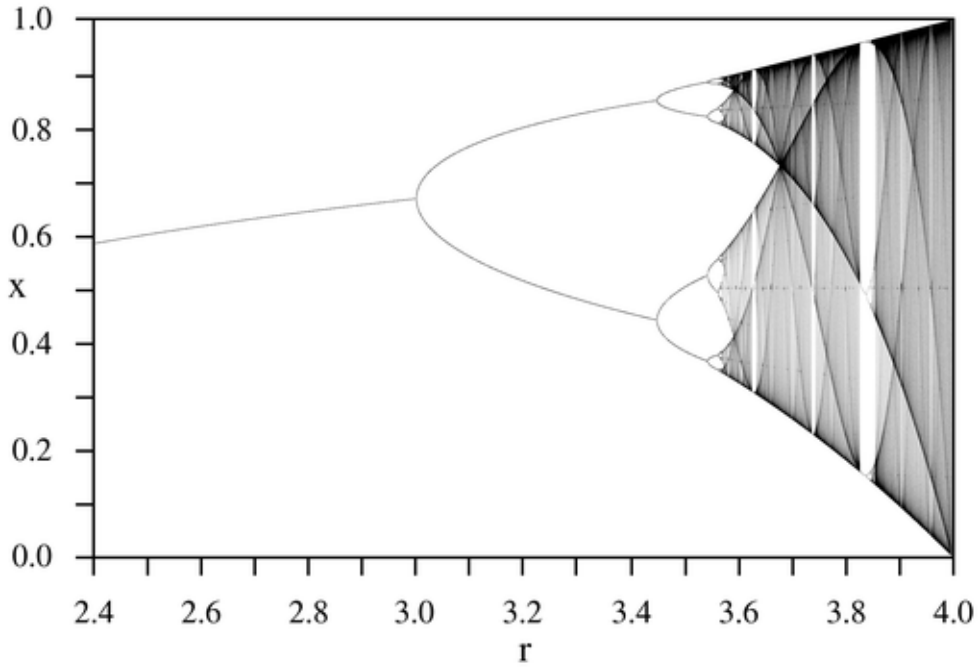


Bild 1.2: Beispiel eines deterministischen Chaos anhand eines einfachen Modells [31]

Genau wie in diesem Beispiel führen kleine Änderungen an den Anfangsbedingungen einer komplexen Simulation zu großen Abweichungen der Endergebnisse. Zum Beispiel kann bei $r = 4$ der Ausgang x zwischen seinem minimalen und maximalen Wert variieren, je nach Anfangsbedingung.

1.2 Zielsetzung

Sicherlich ist die zunehmende Komplexität der Systeme in der Automobilindustrie die Hauptursache für die oben erwähnten Probleme bei der gekoppelten Simulation. Jedoch bietet die Reduzierung des Komplexitätsgrades keine große Abhilfe, denn sie ist schwer realisierbar.

Vielmehr setzen die Berechnungsingenieure auf die Robustheit der Schnittstelle und auf die Kompatibilität der Modelle, die bei mehreren Tools Verwendung finden.

Seit Beginn der Entwicklung der aktiven und passiven elektronischen Sicherheitssysteme Ende der neunziger Jahre stieg der Wunsch nach einem integrierenden Simulationssystem, das die Wirkung dieser regelungstechnischen Eingriffe auf die Dynamik des Fahrzeugs analysiert.

Diese Dissertation hat als Ziel die Verbesserung der Schnittstelle zwischen einem Mehrkörper-Simulationstool, mit dessen Hilfe die Dynamik des Fahrzeugs abgebildet wird, und einem regelungstechnischen Tool, mit dem die Sicherheitssysteme entwickelt werden. Somit können deterministische Ergebnisse aus der Simulation gewährleistet werden. Der Nutzen ergibt sich aber für die allgemeine Verwendung solcher Schnittstellen, unabhängig von den Modellen und deren Teildisziplinen.

1.3 Gliederung

Die Arbeit ist in drei große Abschnitte unterteilt. Zunächst behandeln die Kapitel 2 und Kapitel 3 die Grundlagenforschung. Sie bilden den theoretischen Leitfaden für die Dissertation. Das nächste Kapitel 4 erläutert das Problem in den vorhandenen Konstellationen, während Kapitel 5 und 6 einen Lösungsvorschlag anbieten mit anschließendem Test auf seine Funktionalität.

Im Folgenden wird eine genauere Unterteilung der Arbeit erläutert. Ein Überblick der Beschreibungsmethoden kann bei der Suche nach einer gemeinsamen Abstraktionsebene bei der Tool-Kopplung helfen. Deshalb wird in Kapitel 2 eine Auflistung der bekanntesten Beschreibungsmethoden gegeben mit dem Fokus auf denen, die in den Softwarepaketen ADAMS und MATLAB eingesetzt werden. Weiterhin stellt dieses Kapitel Mehrkörpersysteme wie MSC.ADAMS und blockschaltbildorientierte Tools wie MATLAB/Simulink vor. Der dritte Teil dieses Kapitels gibt einen Überblick über die verschiedenen Integrationsmethoden, die vorwiegend in dieser Arbeit benutzt werden.

Der Einsatz von *Co-Simulationen* oder Gesamtsystemsimulationen in der Industrie ist seit kurzem ein übliches Vorgehen, in wie weit dieser Einsatz jedoch in den Produktentwicklungsprozess integriert ist, wird am Anfang von Kapitel 3 diskutiert. Danach folgt eine kurze Einführung in die Arbeitsweise von „aktuellen“ Gleichungslösern der eingesetzten Tools.

Im Anschluss findet eine Evaluierung der Kopplungsmethoden zwischen den einzelnen Simulationswerkzeugen statt. Dabei wird das Hauptaugenmerk auf die Beziehung zwischen Rechenzeit und Effizienz der Ergebnisse gelegt.

Eine Analyse der vorhandenen Schnittstelle in ihrer bisherigen Konstellation wird in Kapitel 4 durchgeführt. Es werden unter anderem die mathematischen und programmiertechnischen Definitionen wiedergegeben, sowie Einflüsse von verschiedenen Simulationsparametern untersucht. Eingriffsmöglichkeiten zur Verbesserung der Funktionalität der Schnittstelle werden somit lokalisiert und Anforderungen an die Schnittstelle präzise festgelegt.

Gegenstand des 5. Kapitels ist der Lösungsansatz für das in Kapitel 4 beschriebene Event-Problem. Ein Konzept eines Eventhandlers wird entworfen, die Umsetzung wird anschließend detailliert beschrieben und in den Simulationsablauf integriert.

Als logische Folge ist Kapitel 6 für den Test der neu definierten Schnittstelle vorgesehen. Es werden Modellbeispiele mit steigender Komplexität benutzt; angefangen mit einem fremderregten Feder-Masse-System, gefolgt von einem geregelten invertierten Pendel und zum Schluss ein Fahrzeugmodell mit einem ABS-Regler. Die Ergebnisse werden hinsichtlich der Plausibilität der Simulationsergebnisse und des Rechenaufwandes bewertet.

Eine allgemeine Beurteilung der Ergebnisse sowie ein Ausblick auf zukünftige Anwendungen und weiterführende Entwicklungsmöglichkeiten der neu definierten Schnittstelle findet im abschließenden Kapitel 7 statt.

2 Grundlagen

2.1 Einführung: Mechatronik

Jahrelang wurden Entwicklungen in den Bereichen Elektrotechnik, Maschinenbau und Informatik unabhängig voneinander durchgeführt. Doch die zunehmende Kommunikation und Konnektivität zwischen den Komponenten eines Systems, das mit Hilfe dieser verschiedenen Disziplinen entwickelt wurde, führten zur Entstehung einer neuen übergreifenden Disziplin. Somit entstand Ende der neunziger Jahre die Mechatronik, die diese drei bisher weitgehend getrennten Fachbereiche miteinander verschmelzen ließ. [32]

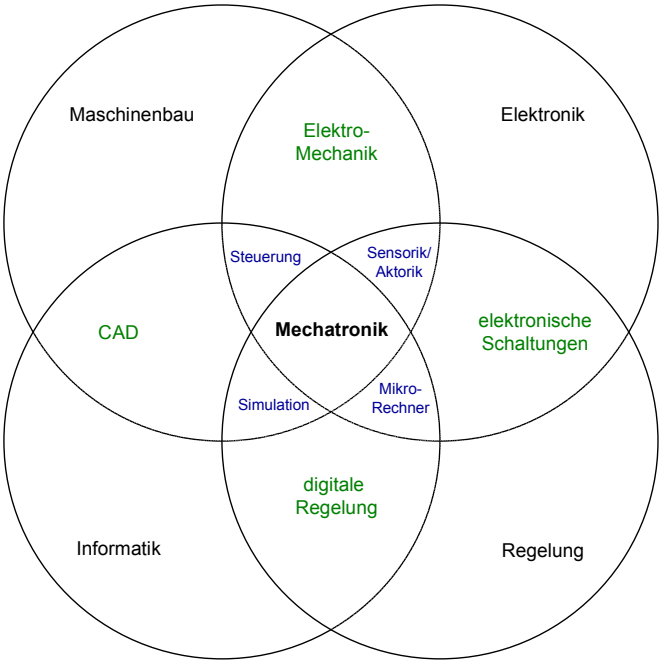


Bild 2.1: Zusammensetzung der Mechatronik aus verschiedenen Disziplinen

Ziel der Mechatronik ist es, Systeme zu entwickeln, die durch ihre Einfachheit gekennzeichnet, zuverlässig, leicht umkonfigurierbar und wirtschaftlich günstig zu realisieren sind.

Meistens interagieren mechatronische Module nach einem regelungstechnischen Plan, der die Dynamik oder das Verhalten des Systems vorher bestimmt und zu den erwünschten Zielen führt. Insbesondere beim Entwurf mechatronischer Systeme entsteht ein Syntheseproblem, dessen technische Umsetzung im Allgemeinen ein fundiertes regelungstechnisches Wissen voraussetzt. Weiterhin sind Kenntnisse aus den Bereichen Sensorik, Aktorik und Datentechnik notwendig. Dabei ist das Hauptziel die Aufbereitung aller Informationen für eine fachübergreifende Verwendung.

In ihrem Buch *Modelling and Simulation of Mechatronic Systems* [29] gehen C. Karnopp, L. Margolis und C. Rosenberg für die Beschreibung der Systeme von Annahmen aus.
Erstens wird mit Hilfe physikalischer oder konzeptbedingter Grenzen ein System als eine vom restlichen Universum[1] trennbare Entität angenommen. Ein Tier, zum Beispiel, kann als ein System angesehen werden, das auf seine Umgebung einwirkt und mit ihr Energie und Informationen austauscht. In diesem Fall sind die Grenzen sowohl physikalischer als auch räumlicher Natur. Dagegen ist ein Flugsicherungssystem ein komplexes, von Menschen entwickeltes System, dessen Umgebung nicht nur aus den physikalischen Gegebenheiten, sondern auch aus den schwankenden Anforderungen des Flugverkehrs besteht. Das Einheitliche in diesen beiden verschiedenen Systemen ist die Fähigkeit zu entscheiden, welche Elemente zum System gehören und was als eine externe Störung oder als eine außerhalb des Systems agierende Steuerung betrachtet werden muss.
Zweitens besteht ein System aus interagierenden Teilen. Bei einem Tier erkennt man Organe mit spezifischen Funktionen. Zum Beispiel dienen die Nerven der Informationsübertragung. Das Flugsicherungssystem besteht allerdings aus Menschen und Maschinen mit entsprechenden Kommunikationsschnittstellen. Da das System in eine übermäßige Zahl von Komponenten zerlegt werden kann, und somit die Analyse durch unnötige Details erschwert wird, ist die Zerlegung des Systems in verschiedener Komponenten als eine Aufgabe zu sehen, die technische Fertigkeiten und eine gewisse Professionalität verlangt.

Wegen den oben genannten Aspekten muss man der Beschreibung der Systeme besonders viel Aufmerksamkeit widmen. Deshalb werden in diesem Kapitel die Beschreibungsmethoden vorgestellt *(siehe 2.2)* und der Unterschied zwischen kausalen und akausalen Beschreibungen wird verdeutlicht. Außerdem wird beim Entwurf des Systems zwischen formalen Methoden wie dem Aufstellen von Differentialgleichungen und nicht-formalen Methoden wie der Benutzung von Simulationssoftware mit einem höheren Abstraktionsgrad unterschieden *(siehe 2.3)*.

Die im Rahmen dieser Arbeit vorwiegend verwendeten Simulationsprogramme und deren

[1] hier ist die gesamte Umgebung des Systems gemeint

theoretische Grundlagen werden in diesem Kapitel 2 vorgestellt. Es werden Mehrkörpersysteme beschrieben und darauf basierend das Tool ADAMS mit seinen Paketen View, Car und Controls *(Unterkapitel 2.5.1, 2.5.2 und 2.5.3)*. Weiterhin beschreibt *das Unterkapitel 2.7* die Software MATLAB/Simulink als eine Applikation einer kausalen blockschaltbildorientierten Beschreibungsform.

2.2 Beschreibungsmittel und Modellbildung

In einem mechatronischen System ist die Systemtheorie ein zentrales Bindeglied zwischen den oben erwähnten Teildisziplinen Mechanik, Elektrotechnik und Informatik. Kern dieser Theorie ist die methodische Beschreibung bzw. Modellbildung des Systems, die den Systementwurf unterstützt und teilweise mitgestaltet. Somit gewinnen Beschreibungsmittel im Laufe der Entwicklung im Automotive-Bereich immer mehr an Bedeutung. Sie tragen dazu bei, komplexe Systeme effizient, kostengünstig und mit möglichst wenigen Fehlern zu entwerfen.

2.2.1 Historischer Überblick

Lange Zeit galten die Systembeschreibungsmittel mathematischer Natur wie die Differentialgleichungen als die einzigen Methoden, mit deren Hilfe dynamische Systeme beschrieben und nachgebildet werden können. Doch die Einführung der Systemtheorie in den 30er Jahren ebnete den Weg für neue Methoden.

Vor allem die Entwicklung elektronischer Systeme und deren steigende Komplexität führten zu neuen Ansätzen bei der Beschreibungsmethodik. Somit entstanden Methoden, die jeweils für bestimmte Systemgruppen geeignet sind.

In [23] findet man eine ausführliche Auflistung der in der Automatisierungstechnik bis 1999 verwendeten Methoden. Zunächst beschränken wir uns hier in diesem Überblick auf die Methoden, die eine markante Rolle im Laufe der Entwicklung gespielt haben. Sie werden in kontinuierliche, diskrete und hybride Gruppen unterteilt.

Beschreibungsmittel für kontinuierliche Systeme

Auf diesem Gebiet hat sich seit Jahren nicht viel getan. Nach wie vor beruhen die meisten Ansätze der Beschreibung rein kontinuierlicher Systeme auf algebraischen Funktionen und

Differentialgleichungen.

- **Differentialgleichungen:**

  > „Wir nennen eine Bestimmungsgleichung für eine Funktion einer unabhängigen Veränderlichen eine gewöhnliche Differentialgleichung, wenn in ihr eine Ableitung der gesuchten Funktion nach der unabhängigen Veränderlichen enthalten ist.“ [40]

  Eine Differentialgleichung ist formal gesehen eine mathematische Gleichung, die mit mindestens einer Ableitung einer Veränderlichen aufgestellt wird. Die höchste Ableitung des Systems gibt die Ordnung der Differentialgleichung an. In der mathematischen Fachliteratur werden je nach Form und Anzahl der Veränderlichen viele Arten von Differentialgleichungen aufgezählt. So wird in [35] und [34] zwischen homogenen, inhomogenen, partiellen [39], stochastischen, linearen, semi-linearen, quasi-linearen, bernoullischen, riccatischen, Abelschen, exakten und vollständig-nichtlinearen Differentialgleichungen unterschieden [37]. Bei der Modellierung dynamischer Systeme kann man auf die Differentialgleichungen nicht verzichten, denn sie stellen meistens die Ausgangsmodelle für die weiteren Beschreibungsformen des Systems dar. Dies begründet die sehr große Zahl an Literatur auf diesem Gebiet.

- **Fourier-Transformation:**
  Da die Beschreibung der Systeme anhand von Differentialgleichungen im Zeitbereich deren Behandlung und weitere Analyse einschränkt, entwickelte der französische Mathematiker Jean Baptiste Fourier 1822 die Fourier-Transformation. Es handelt sich dabei um die Umformung eines Signals in eine endliche Summe von trigonometrischen Funktionen mit allen vorkommenden Frequenzen im jeweiligen Spektrum.
  Speziell in der Regelungstechnik stellt die Fourier-Transformation ein wichtiges Werkzeug dar, das Systeme im Frequenzbereich beschreibt. Somit können wichtige Merkmale des Systems besser als im Zeitbereich beschrieben und für weitere Bearbeitung vorbereitet werden. Die Stabilität des geschlossenen Regelkreises wird dann besser untersucht und für die Reglerauslegung berücksichtigt. [36] [38]

- **Laplace-Transformation:**
  Noch eine wichtigere Rolle stellt die Laplace-Transformation dar, die das System im Bildbereich abbildet. Sie besitzt fast die gleichen Eigenschaften wie die Fourier-Transformation. Weiterhin erlaubt sie in den meisten Fällen die Konvergenz des Bestimmungsintegrals und besitzt eine einfachere Darstellungsform. Ausführlich beschreibt der deutsche Mathematiker Gustav Dötsch in seinem *Handbuch der Laplace-Transformation*

[41] die mathematischen Eigenschaften dieser Transformation und führt dadurch einige Anwendungen zur Lösung vieler Problemstellungen in der Physik und der theoretischen Elektrotechnik zu (siehe [43]). In der regelungstechnischen Literatur findet man eine sehr breite Anwendung dieser Transformation. Sie ist die Grundlage für die **Blockschaltbild**-Darstellung dynamischer Systeme [42]. Mit der BSB[2]-Beschreibungsform hat sich die Aufgabe des Reglerentwurfs sehr vereinfacht. Hinzu kamen andere Beschreibungsformen zum Einsatz, um Stabilität- und Entwurfskriterien festzulegen. Darunter kann man die Wurzelortskurve nennen, die eine grafische Darstellung der Lage der Pol- und Nullstellen der komplexen Führungs-Übertragungsfunktion eines Regelkreises in Abhängigkeit eines Parameters der offenen Kette darstellt.

Im Allgemeinen kann man festhalten, dass die Beschreibungsmittel für kontinuierliche Systeme und die darauf basierenden Tools DGL[3], Fourrier- und Laplace-Transformationen immer als Ausgangspunkt verwenden. So kamen Anfang der 60er Jahre die **Bond Graphen** Zustande, die aber erst durch Karnopp und Rosenberg in den 70ern eine breitere Anwendung in der Automatisierungstechnik gefunden haben [44]. Mit ihrer Hilfe können die Differentialgleichungen dynamischer Systeme grafisch dargestellt werden. Jedoch gab es immer wieder Ansätze, bei denen die dynamischen Systeme nicht explizit durch ihre Differentialgleichungen beschrieben wurden, sondern in einer „gekapselten“ Form, so dass Änderungen in den Modellen einfacher durchzuführen waren.

Beschreibungsmittel für diskrete Systeme

Im Gegensatz zum kontinuierlichen Bereich findet man in der Literatur eine große Anzahl an Beschreibungsmethoden für diskrete Systeme. In 2.2 ist ein Überblick über die am meisten verwendeten Methoden in der Automatisierungstechnik abgebildet, der zum größten Teil aus der Auflistung in [23] stammt.

Küpfmüller legte 1930 den Grundstein für die **Systemtheorie** und stellte die **blockorientierte Beschreibung** in der Nachrichtentechnik vor [45]. In den 50er Jahren führten die Arbeiten von Mealy, Moore, Huffmann und Kleene zur Entstehung der **Automatentheorie** [47], die anhand des aktuellen Zustandes und des Inputs den Folgezustand und den Output ermittelt. Akers und Talanzev machten 1959 wichtige Schritte im Bereich des booleschen Differentialkalküls. Das **BDK**[4] erlebte erst in den 90ern eine Renaissance im Bereich der Steuerungstechnik (siehe beispielsweise [50]); es konnten dann unterschiedliche Ansätze

[2] Blockschaltbild
[3] Differentialgleichungen
[4] Boolescher Differentialkalkül

der ereignisdiskreten Systemtheorie, wie z. B. Automatentheorie in einer einheitlichen und geschlossenen Form behandelt und deren spezifische Vorteile vereinigt werden [48]. 1962 präsentierte der Leipziger Mathematiker C. A. Petri in seiner Dissertation „Kommunikation mit Automaten“ [49] das Konzept der **Petrinetze**. Durch die einfache Darstellung von nebenläufigen Ereignissen verallgemeinern die Petrinetze die Automatentheorie. Bis heute bieten Erweiterungen der Petrinetze ein aktuelles Forschungsthema in der Automatisierungstechnik, wie z. B. die verallgemeinerten stochastischen Petrinetze (**GSPN**[5]) [15]. Mit steigender Komplexität der technischen Systeme erschien Anfang der 70er die **Entscheidungstabelle** [51]. Sie bietet zwar keine technische Verarbeitung aber eine exakte Prüfung der Konsistenz bzw. der Vollständigkeit der Systeme. Angesichts der Wichtigkeit der Ereignisse in diskreten Systemen entstand Mitte der 70er die diskrete Event-Spezifikation (**DEVS**). Anfang der 80er Jahre wurde die **SADT**[6]-Methode entwickelt, bei der sowohl die Aktivitäten als auch die Daten in einem Prozess mit den gleichen Mitteln grafisch beschrieben werden [52]. Zeitgleich beschreibt Chen in [16] Systeme durch Entitäten, Instanzen, Attribute und Relationen mit dem **Entity Relationship Diagram** (ERD). In [53] beschreibt der Autor die **Temporal Logic**-Methode als eine Erweiterung der logischen Operatoren durch eine zeitliche Begrenzung ihrer Gültigkeitsbereiche.
Die rasante Entwicklung im Bereich der Automatisierungstechnik ließ das Interesse an Beschreibungsmitteln wachsen und so kann man in der Literatur dieser Zeit eine große Zahl von Methoden zur Beschreibung diskreter Systeme wiederfinden. Ein Beispiel dafür sind die **Markov-Ketten** in [54], die **Jackson Diagramme** in [55], die **Condition/Event-Systeme** in [17], die **SA/RT**[7] in [56] und die drei ähnlichen **B-, Z-, und VDM-Methoden**, welche die Systeme als einen Verbund abhängiger abstrakter Maschinen darstellen [57] [58] [59]. Eine große Bedeutung gewann die **LOTOS**[8]-Methode für die verteilten Systeme, die dann formal und hierarchisch spezifiziert werden konnten [60]. Außerdem spielt die objektorientierte Modellierungssprache **UML**[9] eine sehr wichtige Rolle und findet eine breite Verwendung in verschiedenen technischen Bereichen. Diese Sprache wurde von Booch, Rumbaugh und Jacobson entwickelt und später von der *Object Management Group*[10] standardisiert. UML stellt einen wichtigen Schritt zur Entstehung einer einheitlichen Beschreibungsumgebung innerhalb eines dynamischen Systems dar und wird daher als eine wichtige Modellierungssprache in der Forschung und in der Industrie angesehen [62].

[5] Generalized Stochastic Petri Net
[6] Structured Analysis and Design Technique
[7] Structured Analysis/Real Time
[8] Language Of Temporal Ordering Specification
[9] Unified Modeling Language
[10] Ein Konsortium in der Computer-Industrie, das Standards in der Branche definiert

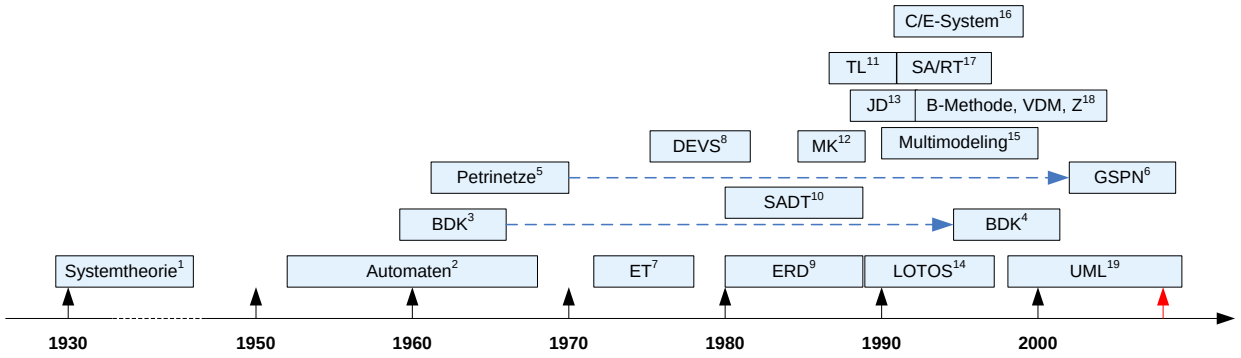


Bild 2.2: Historische Entwicklung der formalen System-Beschreibungsmittel. [1]Küpfmüller 1930; [2]Mealy 1955, Moore 1956 und Kleene 1956; [3]Boolescher Differentialkalkül: Akers und Talanzev 1959; [4]Renaissance der BDK: Kowalewski 1996; [5]Petri 1962; [6]Generalised Stochastic Petri Net: Erweiterung der Petrinetze; [7]Entscheidungstabellen: Dressler 1973; [8]Discrete Event Specification: Zeigler 1976, Thomas 1995; [9]Entity Relationship Diagramme: Jordan und Machesky 1990; [10]Structured Analysis and Design Technique: Connor 1980; [11]Temporale Logik: Krüger 1987; [12]Markov-Ketten: Marsan 1986; [13]Jackson Diagramme: Sutcliffe 1988; [14]Language Of Temporal Ordering Specification: Van Eijk 1989; [15]Multimodeling: Fishwick 1993; [16]Condition/Event-Systeme: Sreenivas Krogh 1991; [17]Structured Analysis/Real Time: Raasch 1991; [18]B-Methode: Abrial 1996, VDM: Harry 1996, Z: Lightfoot 1991; [19]Unified Modeling Language: Burkhardt 1998;

Beschreibungsmittel für hybride Systeme

Die entwickelten Beschreibungsmittel für die hybriden Systeme, die sowohl aus kontinuierlichen als auch diskreten Teilsystemen bestehen, sind in der Regel Erweiterungen der bereits existierenden Mittel für kontinuierliche bzw. diskrete Systeme. So wurden in den 90er Jahren verschiedene Ansätze entwickelt. Ein Teil davon ist in der Tabelle 2.1 aus [23] aufgelistet.

2.3 Simulation und virtuelles Prototyping

2.3.1 Simulation

Im Zuge der Entwicklung eines Prozesses bzw. eines Systems wird man mit der Aufgabe konfrontiert, ein Modell für das Vorhaben zu erstellen. Somit können Aussagen über das Verhalten des zu implementierenden Systems getroffen werden.
Nach der VDI-Richtlinie ist die Simulation:

Tabelle 2.1: *Zusammenstellung aktueller Ansätze zur Beschreibung hybrider Systeme ([23])*

Ansatz	Entwickler	Grundmodell	Zweck	Einschränkungen
Hybride Petrinetze	David und Alla	S/T-Netze	Modellbildung und Analyse	strukturell
Hybride Objekt-Netze	Drath	S/T-Netze	Simulation	strukturell
Hybrid flow Nets	Flaus	S/T-Netze	Synthese	strukturell
Hybride gefärbte Netze	Decknatel	gefärbte Netze	Modellbildung	keine
Geschaltete DGLn	Nenninger (HDM)/Müller	(SI)PN / DGL	Modellbildung / Analyse	keine
Hybride Automaten	Alur et AL.	Automaten	Modellbildung und Analyse	keine
Hybride B/E-Systeme	Krogh	B/E-Systeme und Automaten	Modellbildung und Analyse	keine
Diskrete Abstraktion	Raisch und O'Young	endliche Automaten	Synthese	keine

> „. . . das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind." [20]

Der Schwierigkeitsgrad dieser Aufgabe hängt von sehr vielen Faktoren ab. Dazu zählen folgende Punkte:

- **Art des zu simulierenden Systems:**
  Nach der in 2.1 erläuterten Definition können die Systeme in der Automatisierungstechnik entsprechend ihrer Merkmale und Eigenschaften klassifiziert werden. Speziell in der Regelungstechnik erlauben in der Regel die mathematischen Modelle des Systems eine genaue Zuordnung zu der jeweiligen Klasse. Die Tabelle 2.2 listet einige Klassen von Systemen auf. Es ist also sehr wichtig, das System vor der Modellierung zu klassifizieren, um die Modellierungsziele nicht aus den Augen zu verlieren. Weiterhin erlaubt die Klassifikation die richtige Auswahl der Beschreibungsmethode und folglich des Simulationswerkzeuges.
  Insgesamt werden hier fünf Kriterien aufgelistet, die die Systemeigenschaften beschreiben. Zum Beispiel kann man anhand der Natur der Systemgrößen, abhängig davon ob sie kontinuierlich oder diskret sind, das System entweder mit Differential- oder Diffe-

Tabelle 2.2: System Klassifizierung

Kriterium	**Eigenschaft: Beschreibung**	
Wechselwirkung mit der Umgebung	offen: vorhanden	abgeschlossen: keine
Systemgrößen	dynamisch: zeitlich veränderlich	statisch: zeitlich unveränderlich
Systemgrößen	kontinuierlich: kontinuierliche Änderung in unendlich kleinen Zeitabschnitten	diskret: Änderung in endlichen Zeitabschnitten
Reproduzierbarkeit der Ergebnisse	determiniert: ja, bei identischen Bedingungen	stochastisch: nein, auch bei identischen Bedingungen sind die Ergebnisse nur Annäherungen
Gleichgewicht	stabil: vorhanden bei kleinen Änderungen	instabil: nicht mehr vorhanden bei kleinen Änderungen

renzengleichungen beschreiben.

- **Tiefe der Modellierung:**
  Im Laufe der technischen Entwicklung nahmen abstrakte Modelle im Produktlebenszyklus eine unabdingbare Position ein. Ein abstraktes Abbild eines Systems löst teilweise die physikalischen Modelle bei ihren Aufgaben ab. Ist das System physikalisch zu komplex, um es vollständig zu beschreiben, dann können Teilmodelle entstehen, die nur bestimmte Eigenschaften des Systems nachbilden. Man spricht auch von Modellreduktion auf die wesentlichen Wechselwirkungen des Systems. Modelle können dann je nach Aufgaben in Klassen unterteilt werden, z.B. Beschreibungsmodelle zur Veranschaulichung der internen Abläufe innerhalb des Systems, Erklärungsmodelle zur physikalischen Erfassung der internen Abläufe oder Entscheidungsmodelle zur möglichen Vorhersage der Systementwicklung.

- **Übertragbarkeit der Ergebnisse auf die Wirklichkeit:**
  Da das Ziel einer Simulation in erster Linie die Bildung einer Prognose innerhalb der

Modellebene ist, sollte das abstrakte Modell möglichst das reale Systemverhalten nachahmen. Besonders wichtig wird diese Nachahmung für die quantitativen Computersimulationen, bei denen das zukünftige Verhalten des Systems vorhergesagt und berechnet wird. So muss ein Kompromiss erzielt werden zwischen der Vereinfachung der Modelle auf einer Seite und einem tiefen Detaillierungsgrad zur Erfüllung der Simulationsziele auf der anderen Seite.

2.3.2 Virtuelles Prototyping

Im Zuge der Entwicklung industrieller Produkte wurden in der Vergangenheit meistens Hardware und Software getrennt voneinander entwickelt. Durch diese Vorgehensweise wurden Entwurfs- und Funktionsfehler erst sehr spät entdeckt. Die Beseitigung dieser Fehler erforderte bislang teure und zeitaufwendige Änderungen sowohl an den Hardware- als auch an den *Softwaredesigns*. Unter Umständen waren einige Iterationen dieses Prozesses notwendig.

Heutzutage fließen die Entwicklungsphasen von Hardware und Software immer mehr und mehr in einander über. Es entsteht somit eine virtuelle Entwicklungsmethodik, bei der die meisten Entwicklungsphasen mittels Simulationen virtuell abgebildet werden. Dieser Prozess wird als *virtuelles Prototyping* bezeichnet.

In [64] beschreiben Thomas Barth und Manfred Grauer das virtuelle Prototyping wie folgt:

> „Unter **virtuellem Prototyping** wird die Gesamtheit der Techniken verstanden, die notwendig sind, um die Produktentwicklung weitgehend computerunterstützt durchführen zu können.“

Neben der positiven Auswirkung auf den wirtschaftlichen Aspekt beim Entwicklungsprozess erlaubt das virtuelle Prototyping die Entwicklung einer robusten Software, die mit den klassischen Entwicklungsmethoden nicht realisierbar wäre.

2.3.3 Digitale Fabrik

Während sich das virtuelle Prototyping auf einzelne Systemteile oder auf die Gesamtheit eines gut definierten Produktes bezieht, benutzt man zunehmend den Begriff „digitale Fabrik“, der sich auf die Simulation der gesamten Fabrik bezieht. Im VDI-Arbeitskreis [63] wird der Begriff so definiert:

„Die **Digitale Fabrik** ist der Oberbegriff für ein umfassendes Netzwerk von digitalen Modellen und Methoden, u. a. der Simulation und 3D-Visualisierung. Ihr Zweck ist die ganzheitliche Planung, Realisierung, Steuerung und laufende Verbesserung aller wesentlichen Fabrikprozesse und -ressourcen in Verbindung mit dem Produkt."

2.3.4 Grenzen der Simulation

Trotz zahlreicher Vorteile kann die Simulation nicht alle Probleme im Produkt-Entwicklungsprozess lösen. Dies ist vor allem auf die Grenzen der Modellierung und der Simulationen zurückzuführen. Die folgenden Aspekte definieren einige dieser Grenzen:

- **Knappheit der Mittel:**
  Energie in Form von Rechenkapazität, Zeit oder Geld sind immer begrenzt und dürfen bestimmte Schwellen nicht überschreiten. Daher muss eine Simulation eine wirtschaftlich günstige Lösung darstellen, um daraus einen Nutzen beim Entwicklungsprozess zu ziehen. Folglich soll das Modell so einfach wie möglich aufgestellt werden, was wiederum zu weniger effizienten Simulationsergebnissen führen kann.

- **Vereinfachungsfehler:**
  Um die Knappheit der Mittel unter Kontrolle zu halten, kann man die Simulationsmodelle vereinfachen. Doch diese Vereinfachung führt besonders bei quantitativen Modellen zu Fehlern.

- **Übertragbarkeit der Parameterbereiche auf die Realität:**
  Meistens konzentriert sich der Entwickler bei der Modellierung auf einen bestimmten Bereich, z. B. durch Linearisierung am Arbeitspunkt. Wird das System in einem anderen Bereich betrieben, so geben die Simulationsergebnisse keinen Aufschluss über das reale Verhalten des Systems in diesem Bereich. Abhilfe könnte die Verifikation der Modelle vor jedem neuen Anwendungsfall schaffen.

- **Ungenauigkeit der Ausgangsdaten:**
  Die Modelle werden meistens anhand von Messdaten gebildet. Jedoch sind Messungen meistens fehlerhaft. Dies führt zu verfälschten Modellen.

- **Rechentechnik:**
  Bei Computersimulationen spielt die Auswahl des Rechenverfahrens eine große Rolle bezüglich Effizienz und Robustheit der Ergebnisse.

- **Rechenfehler:**
  In digitalen Simulationen arbeitet der Rechenalgorithmus immer in endlichen Zeitinkrementen. Je nach Größe dieser Inkremente muss man mit einem Rundungsfehler rechnen, der unter Umständen akkumuliert wird.

Diese Aspekte müssen sowohl bei der Modellierung als auch bei der Analyse der Simulationsergebnisse mit berücksichtigt werden. So spielt die Rechentechnik bei der Simulation von Mehrkörpersystemen, die im folgenden Unterkapitel beschrieben werden, eine große Rolle bei der Bewertung der Simulationsergebnisse.

2.4 Mehrkörpersimulationen

2.4.1 Einführung

Die Mehrkörpersimulation ist eine junge Disziplin. Sie wurde neben den FEM-Systemen für die Analyse mechanischer Strukturen entwickelt. Die rasante Entwicklung dieser Disziplin ist der großen technischen Verbreitung leistungsfähiger Rechnersysteme mit leicht bedienbaren, intelligenten und aussagekräftigen Programmen zu verdanken. In Fachkreisen spricht man vor allem im mechatronischen Bereich von *rechnerorientierter Darstellung der Mechanik*.

Mehrkörpersysteme und deren Simulationen können als allgemeine Methoden zur Formulierung und Lösung der Bewegung mechanischer Systeme aufgefasst werden.

Eine Mehrkörpersimulation ist die Simulation von Mehrkörpersystemen, die die physikalische Beschreibung der technischen Systeme durch Körper und deren Verbindungselemente darstellt. Ein Mehrkörpersystem kann wie folgt definiert werden:

Definition:

> Ein Mehrkörpersystem ist eine Menge von miteinander verbundenen, starren Körpern, die eine mit diesen Bindungen verträgliche Bewegung zueinander ausführen. Man unterscheidet holonome (Lagegrößen einschränkende), nichtholonome (Geschwindigkeitsgrößen einschränkende), skleronome (zeitunabhängige) und rheonome (zeitabhängige) Bindungen. Man bezeichnet die durch Koppelelemente verursachten Kräfte und die Gewichtskräfte als *eingeprägte* Kräfte, die durch Bindungen verursachten Kräfte als *Reaktionskräfte*. [65]

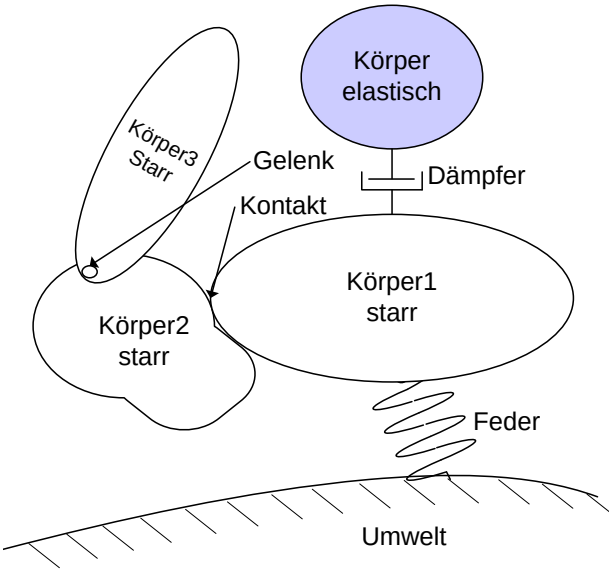


Bild 2.3: Prinzip eines MKS

Mit Mehrkörpersystemen können also Verschiebungen und Verdrehungen von massebehafteten Körpern gut beschrieben werden. Hierbei sollten die einzelnen Körper keinen großen Verformungen unterliegen. Das Prinzip eines Mehrkörpersystemmodells ist die Idealisierung realer Systemstrukturen durch starre Teile, die über eine gute Bindung zu den Teilen selbst und zur Umgebung verfügen. Der größte Vorteil der Mehrkörperbeschreibung liegt in der leichten Ableitung differenzial-algebraischer Gleichungen aus der nach der Abbildung 2.3 strukturierten grafischen Beschreibung; somit können Fehler bei der Aufstellung der Systemgleichungen minimiert oder vollständig vermieden werden. [22]

2.4.2 Bestandteile eines MKS

In der ersten Linie ist ein MKS ein Modell eines realen Systems. Das Modell enthält in der MKS-Logik Körper, die starr oder elastisch sein können, und Verbindungselemente, die die Verbindungen zwischen den einzelnen Körpern untereinander bzw. die Verbindungen von Körpern zur Umwelt definieren. Diese Struktur ist im Organigramm 2.4 beschrieben.

2.4.3 Aufgaben und Ziele einer Mehrkörpersimulation

Mehrkörpersimulationen können zu einer Ansammlung allgemeiner Methoden zur Analyse, Formulierung und Lösung der Bewegungsgleichungen mechanischer Systeme zusammenge-

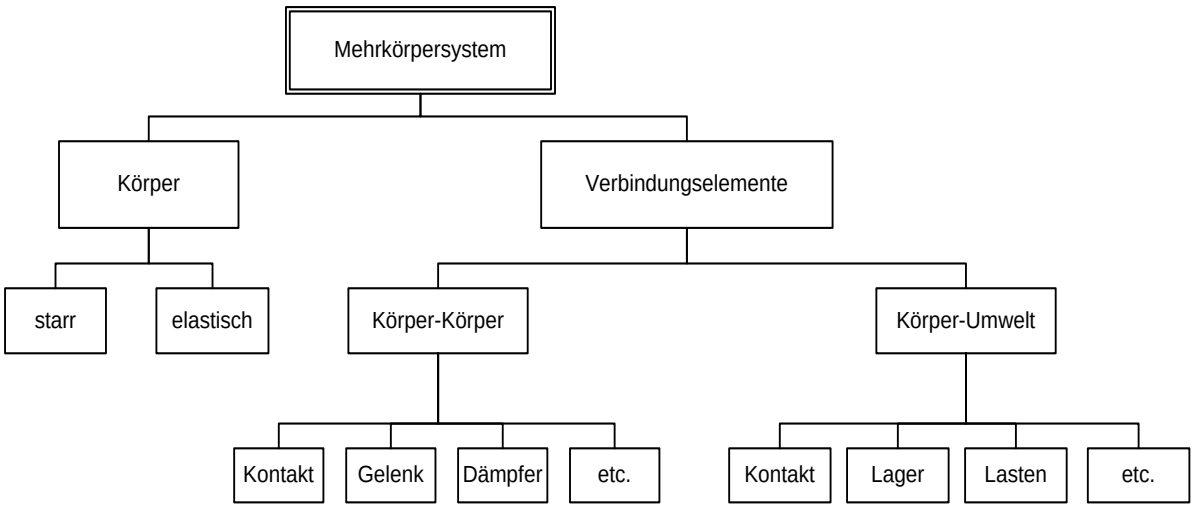


Bild 2.4: Struktur eines MKS

fasst werden. Angesichts der heutigen Entwicklung und Verbreitung leistungsfähiger Rechner und leistungsfähiger Programme sind die oben erwähnten Aufgaben fast ausschließlich numerisch lösbar. Diese Analysen sind in vier Kategorien zu unterteilen:

Kinematik

Es handelt sich hierbei um die Analyse der Bewegung des mechanischen Systems im Raum beschrieben durch die Größen Weg $s(t)$ (Änderung der Ortskoordinate), Geschwindigkeit $v(t)$ und Beschleunigung $a(t)$, ohne Betrachtung der Ursachen dieser Bewegung wie Kräfte oder Momente.

Dynamik

Im Gegensatz zur Kinematik, wird hier die Wirkung der äußeren Kräfte und Momente auf die Bewegung des Systems betrachtet. Zu diesen Wirkungen zählen Kräfte, die von außen wirken wie z.B. die Schwerkraft. Weiterhin sind Kräfte zu berücksichtigen, die aus der Relativbewegung des Körpers oder seiner Unterteile entstehen. Darüber hinaus werden die aus einer Regelung resultierenden Kräfte addiert. In diesem Fall beschreiben Differentialgleichungen und algebraische Zwangsbedingungen das dynamische System.
Ist die Bewegung des Systems anhand der Bewegung von einzelnen Körperteilen beschrieben und die daraus berechneten Positionen, Geschwindigkeiten und Beschleunigungen in die Bewegungsgleichung der Dynamik eingesetzt, so spricht man von inverser Dynamik-Analyse.

Eigenanalysen

Hierbei werden die Eigenfrequenzen und Eigenwerte berechnet. Somit können Resonanz- und Stabilitätsanalysen durchgeführt werden.

Regelung

Im Rahmen der MK-Simulationsprogramme können Parameter und Struktur des Systems identifiziert und anschließend ggf. optimiert werden. Daraufhin können Regelkonzepte entworfen und im Gesamtsystem mit simuliert werden.

2.4.4 Ablauf einer Mehrkörpersimulation

Die Abbildung 2.5 beschreibt, wie aus den Problemstellungen eines mechanischen Systems das MK-Modell mit den entsprechenden Bewegungsgleichungen gebildet, validiert, optimiert und anschließend simuliert wird.

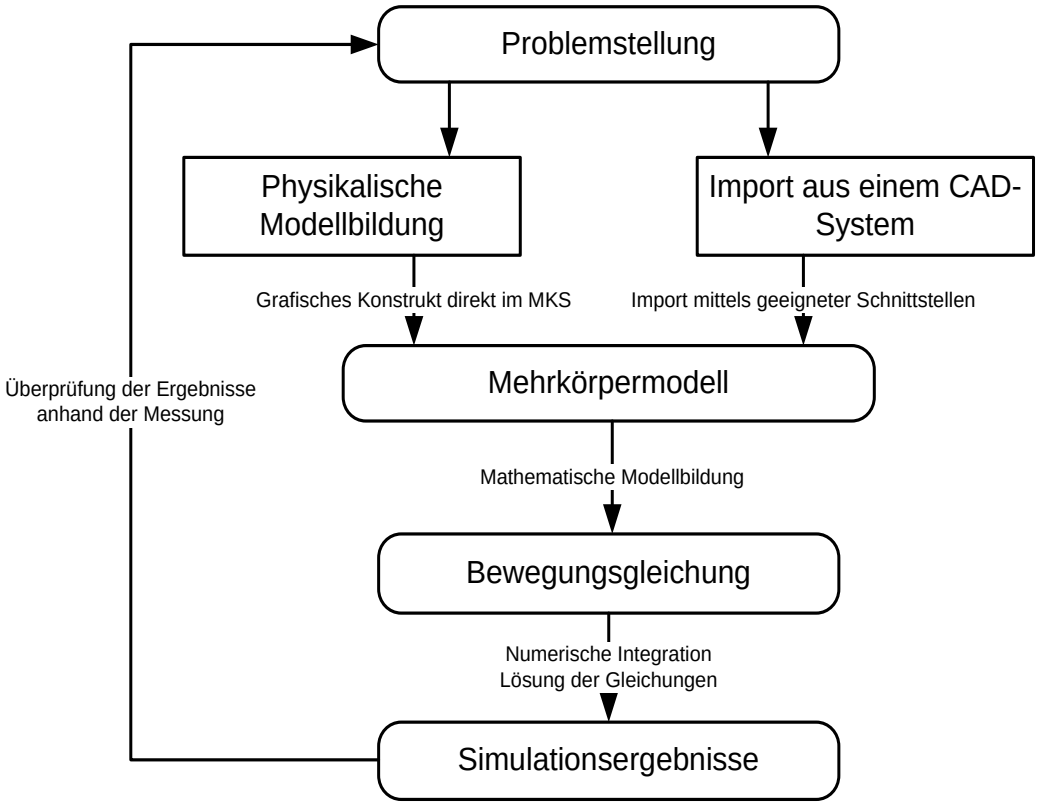


Bild 2.5: Ablauf einer Mehrkörpersimulation

2.4.5 Bewegungsgleichungen

In [66] ist die allgemeine Bewegung eines MKS durch die folgenden drei Gleichungen beschrieben:

$$\begin{aligned} \dot{q} &= Tu \\ M\dot{q} &= h + W\lambda \\ (q, u, \lambda, t) &\in \aleph \end{aligned}$$

dabei sind die oben erwähnten Größen so definiert:

- q: generalisierte Lagekoordinaten
- u: generalisierte Geschwindigkeitskoordinaten
- $T = T(q)$: Abbildungmatrix
- $M = M(q)$: generalisierte Massenmatrix, symmetrisch positiv definit, enthält Trägheitseigenschaften aller Körper
- $h = h(q, u, t)$: Vektor der generalisierten inneren und äußeren Kräfte sowie aller gyroskopischen Kräfte
- λ: generalisierte Kräfteparameter
- $W = W(q)$: Matrix der Kräfterichtungen
- $\aleph$: Menge der Nebenbedingungen in Form von Gleichungen und Ungleichungen

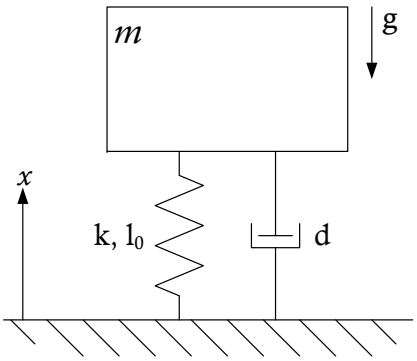


Bild 2.6: Beispiel eines Einmassenschwingers

Beispiel: Bewegungsgleichungen eines Einmassenschwingers Angenommen wird ein Einmassenschwinger nach der Abbildung 2.6 mit der Auslenkung x, der Dämpfung d, der Federkonstante k, der Federlänge l_0 und der Masse m. Die Bewegungsgleichungen sehen demnach wie folgt aus:

$$\begin{aligned} q &= (x) \\ u &= (v) \\ T &= (1) \\ M &= (m); \\ h &= (-mg - c(x - l_0) - dv); \\ W &\equiv 0; \\ \lambda &\equiv 0; \\ \aleph &= \{\} \end{aligned}$$

In einer Mehrkörpersimulationssoftware werden diese Gleichungen aus der Grafik im Hintergrund aufgestellt. Daraufhin werden numerische Integrationsverfahren angewandt, um eine Lösung in einem bestimmten Zeitrahmen zu errechnen und gegebenenfalls zu visualisieren.

2.5 MKS mit MSC.ADAMS

ADAMS (Automatic Dynamic Analysis of Mechanical Systems) ist eine umfangreiche Implementierung der Mehrkörpersimulation. Grundprinzip der Arbeit mit ADAMS ist die grafische Erstellung eines mechanischen Systems ausgehend von den Körpern. Anschließend erfolgt die Definition aller Verbindungsformen dieser Körper miteinander. Daraufhin erstellt die Software im Hintergrund für den Anwender unsichtbar die Bewegungsgleichungen in vektorieller Schreibweise. Die Simulation des Systems basiert auf der numerischen Lösung der Bewegungsgleichungen innerhalb des Simulationsintervalls. Die Ergebnisse umfassen alle im System enthaltenen Variablen, welche in Form von Plots oder einer zur Berechnung simultanen grafischen 3D-Bewegungsdarstellung dargestellt werden.

Das Werkzeug ADAMS besteht aus mehreren Paketen. Einige sind Grundbausteine, mit deren Hilfe der Systemaufbau und anschließend die Berechnung der Bewegungsgleichungen in der Software möglich ist. Dazu zählen die Pakete ADAMS.View, das als GUI (Graphical User Interface) für den Aufbau des mechanischen Systems fungiert, und ADAMS.solver,

mit dem die Berechnung der Lösung des aufgestellten Systems möglich ist. Allgemeine Ergänzungen für Animationen, Ergebnisspeicherung und -bearbeitung sind durch Tools wie ADAMS.Postprocessing gesichert. Andere Module wie Car, Tire, Drive, Vehicle und Rail sind optional und dienen der Simulation fahrzeugspezifischer mechanischer Systeme. Für die Lösung anderer Aufgaben aus anderen Disziplinen stehen andere Pakete zur Verfügung. Beispielsweise sieht ADAMS für die regelungstechnischen Aufgaben das Paket ADAMS.control vor. Gleichzeitig dient das letztere als eine Schnittstelle zu anderen regelungstechnischen Werkzeugen. Weiterhin erstreckt sich der Datenaustausch mit anderen Werkzeugen über verschiedene Bereiche wie FEM (Finite Elemente Methoden) und CAD-Software (Computer Aided Design) hinaus.

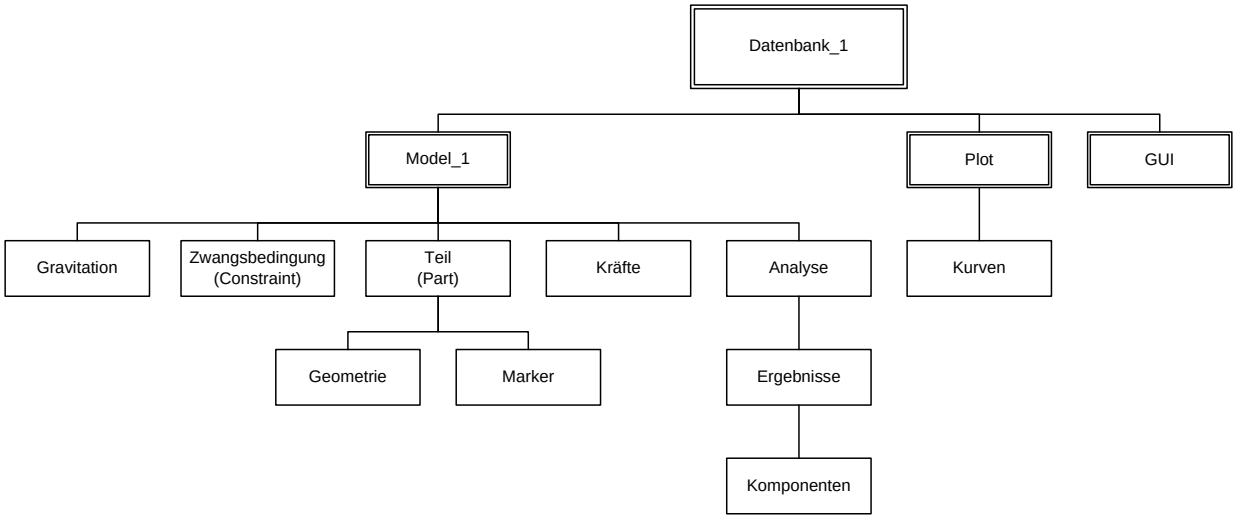


Bild 2.7: Hierarchie einer Datenbank in ADAMS

Die Abbildung 2.7 zeigt, wie die Datenstruktur in ADAMS gebaut ist. Drei Komponenten haben Zugriff auf die Datenbank der Software: das vom Anwender erstellte Modell, eine *Plot*-Einheit zur Darstellung der Ergebnisse nach der Simulation und die grafischen Oberflächen (GUI) zur Bedienung des Programms. Das Modell wird hierbei nach dem MKS-Prinzip aus Teilen (*Parts*), Kräften und Zwangsbedingungen (*Constraints*) gebaut. Diese Struktur wird von den Paketen der Software verwendet, die im Folgenden näher erklärt werden.

2.5.1 Das Paket ADAMS/View

ADAMS/View zählt zu den Kernpaketen in ADAMS. Die ersten Schritte der Modellierung werden in diesem Paket durchgeführt. Diese Schritte spiegeln den gleichen Verlauf wie bei

der physikalischen Prototypherstellung wieder. Insgesamt besteht dieser Prozess aus vier Phasen:

Aufbau des Modells

In dieser Phase werden Teile (Parts) aufgebaut, die den hinzugefügten Zwangsbedingungen und Bewegungen unterstellt sind. Zum Schluss dieser Phase werden Kräfte und Momente dazu addiert, welche die Bewegungen der Teilsysteme direkt induzieren oder indirekt beeinflussen.

Test

Nach dem Aufbau der Teile, deren Verbindungselementen und den darauf wirkenden Kräften und Momenten, werden Ausgangsgrößen für die Ergebnisse definiert. Dazu zählen neben den Standardgrößen wie Wege, Geschwindigkeiten, und Beschleunigungen auch Messungen, die vom Anwender definiert sind, wie z.B. der Abstand zwischen zwei Teilen. Im Anschluss wird die Simulation ausgeführt. Ziel ist die Überprüfung der Funktionalität, der Zuverlässigkeit und der charakteristischen Leistungen des Systems. Ferner kann man den Einfluss der Anfangsbedingungen auf das System näher betrachten.

Überprüfung

Der Anwender kann das Modell in zwei Hinsichten überprüfen. Als erstes sind die Animationen durch das Tool ADAMS/Postprocessing sehr gut steuerbar. Somit können bestimmte Bereiche der Simulation näher betrachtet werden. Zweitens können Daten aus einem physikalischen Prototypen des Systems in View importiert werden. Anschließend ist ein Vergleich möglich, mit dessen Hilfe die Exaktheit des Modells bzw. der Simulation überprüft wird.

Aufbesserung

Zum Schluss kann das Modell verfeinert werden. Komplexere Zusammenhänge wie Reibungen zwischen Körpern und Regelungssystemen in Form von linearen oder allgemeinen Zustandsgleichungen können so addiert werden. Weiterhin können starre Körper durch flexible Körper ersetzt werden. In der Praxis werden jedoch die oben erwähnten Schritte nicht direkt für das gesamte Modell angewandt. Sie werden für einzelne, kleine Teilsysteme durchgeführt, so dass Fehlerquellen leichter einzugrenzen sind.

2.5.2 Einführung in die Umgebung des Pakets ADAMS/Car

Das Paket ADAMS/Car ist eine spezialisierte Umgebung für die Modellierung von Fahrzeugen. Erstrebenswert ist auch in diesem Fall die Erstellung eines virtuellen Prototyps, der möglichst dem realen Prototyp ähnelt und somit seine Aufgaben zum größten Teil übernehmen kann.
ADAMS/Car gehört zu den *template-based* Produkten, die mit vordefinierten Standardblöcken versehen sind. Diese bis auf die spezifischen Parameter vormodellierten Teile erlauben einen schnelleren Prototypenaufbau und reduzieren mögliche Modellierungsfehler. So entsteht eine volle parametrische Modellierungstechnik, bei der die Änderung einer Entität die Änderung aller von ihr abhängigen Komponenten hervorruft. Nach dieser Einstellung werden in ADAMS/Car die Subsysteme, wie vordere Federung, hintere Federung, Lenksystem, Karosserie, usw. zu einem gesamten Fahrzeugmodell (*Assembly*) zusammengefügt. Diese Subsysteme werden auf der Basis von *Templates* gebaut. Die *Templates* können nicht in der Standard-Anwendung der Software geändert werden. Dazu muss man in den *Expert*-Modus wechseln. Änderungen können daher in diesem Modell schneller und sicherer im Standard-Modus durchgeführt werden; während der *Expert*-Modus für grundlegende Erneuerungen in den Subsystemen benutzt wird.

Um ein mit ADAMS/Solver analysierbares System aufzubauen, muss ein *Assembly* neben den üblichen Subsystemen eines Fahrzeugs einen Prüfstand[11] enthalten. In ADAMS/Car können beispielsweise die Subsysteme Lenkung, vordere Federung, und Federungsprüfstand zu einem *Assembly* einer Federung zusammengefasst werden, dessen kinematisches Verhalten analysiert wird.

Die Abbildung 2.8 zeigt die allgemeine Struktur einer Modell-Datenbank in ADAMS/Car. Dabei sind die Kommunikatoren zwischen den Subsystemen, dem Prüfstand und den *Templates* übersichtshalber nicht explizit gezeichnet.

2.5.3 Einführung in die Umgebung des Pakets ADAMS/Controls

Das Paket ADAMS/Control ist ein Zusatzmodul für ADAMS/Car, View, Solver und andere Pakete. Als Plug-in ermöglicht es die Anbindung ausgeklügelter Regelungen am Modell.

[11] test rig

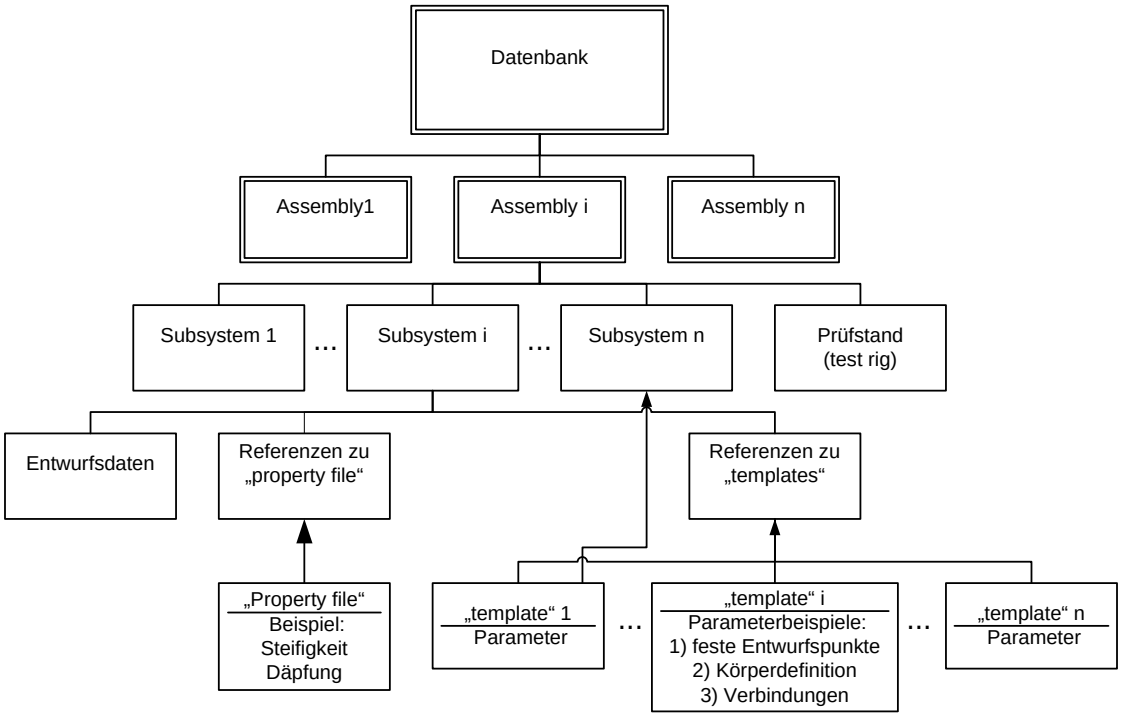


Bild 2.8: Modellstruktur in ADAMS/Car

Vorteilhaft ist vor allem die Anbindung von Regelungen aus anderer Software wie MATLAB/Simulink, die im Unterkapitel 2.7 vorgestellt wird. Diese Anbindung bietet dann drei Arbeitswege:

- Simulation des Modells mit sämtlicher Regelung innerhalb der Regelungssoftware
- Simulation und Analyse des gesamten Modells in ADAMS
- Lösung der Regelungsgleichungen anhand der Regelungssoftware und Lösung der mechanischen Gleichungen mit dem ADAMS/Solver, das im folgenden Unterkapitel kurz erläutert wird.

Um die Regelung am mechanischen System anzubinden, wird ein vier-Schritt-Prozess verwendet, der im Folgenden beschrieben wird:

1. **Modellentwurf:**
   Als erstes muss ein mechanisches Modell in ADAMS entworfen oder importiert werden. Das Modell muss vollständig sein.

2. **Identifizierung der Ein- und Ausgänge:**
   Im zweiten Schritt sind Ein- und Ausgänge des ADAMS-Modells zu definieren. Die

Ausgänge gehen als Eingänge in die Regelung ein. Ebenso sind Eingänge des mechanischen Modells als Ausgänge der Regelung zu betrachten. Somit schließt sich der Regelkreis, siehe Abbildung 2.9.

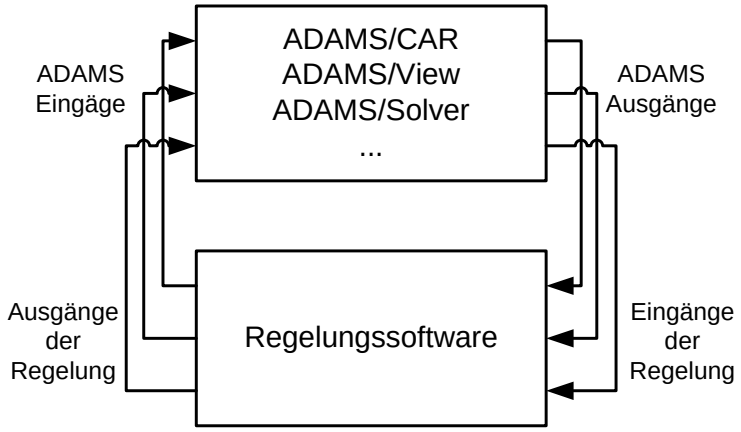


Bild 2.9: ADAMS mit einer Regelungssoftware in einem geschlossenen Regelkreis

3. **Reglerentwurf:**
   Mit den vordefinierten Ein- und Ausgängen werden die Regelungen in der Regelungssoftware wie MATLAB/Simulink entworfen. Dabei wird die mechanische Strecke als ein Blockdiagramm in die Regelungssoftware importiert. Alternativ dazu kann die Regelung beispielsweise mit dem Real-Time-Workshop nach ADAMS exportiert werden.

4. **Simulation des Modells:**
   Im letzten Schritt wird das kombinierte Modell aus dem mechanischen Teil und der Regelung als ein geschlossener Regelkreis simuliert.

2.6 Einführung in Matlab

In erster Linie ist MATLAB ein Software-Produkt, dessen Hauptaufgabe die technische Berechnung von Systemen jeder Art ist. Weiterhin ist MATLAB auch eine Programmiersprache mit eigenen Vorteilen und Nachteilen gegenüber anderen Sprachen. Außerdem bietet MATLAB gute Visualisierungsmöglichkeiten.
Historisch gesehen wurde MATLAB entwickelt, um einen einfachen Zugang zur Matrix-Software aus den Projekten LINPACK und EISPACK zu verschaffen. Daher stammt der Name: **MAT**rix **LAB**oratory. MATLAB zeichnet sich durch eine Reihe von spezifischen

Lösungen für bestimmte Problemstellungen oder bestimmte Gebiete aus. Diese Softwareteile werden als *Toolboxes* bezeichnet. Sie sind eine Ansammlung von MATLAB-Programmen oder Funktionen, *m-files* genannt. Sie umschließen einen großen Bereich von Problemklassen, z.B. Signalverarbeitung, Regelung, neuronale Netzwerke usw.

In seiner Aufbaustruktur wird MATLAB in fünf Teile unterteilt:

- **Entwicklungsumgebung:**
  Dies ist eine Reihe von Werkzeugen, die es ermöglichen, die MATLAB-Funktionen und Dateien zu verwenden. Die meisten dieser Werkzeuge sind GUIs.
- **Bibliotheken der mathematischen Funktionen:**
  Es handelt sich hier um eine Sammlung von Berechnungsalgorithmen, ausgehend von einfachen Funktionen wie Additionen und Subtraktionen bis hin zu komplexen Algorithmen wie die Berechnung der Inversen einer Matrix und ihrer Eigenwerte oder die Berechnung der Fast-Fourier-Transformation.
- **MATLAB als Programmiersprache:**
  MATLAB ist eine höhere Programmiersprache mit großen Vorteilen in der Bearbeitung von Vektoren und Matrizen. Weiterhin bietet sie einige Vorteile der objektorientierten Programmierung.
- **Grafik in MATLAB:**
  Es können Vektoren und Matrizen in Form von Grafiken dargestellt werden. Mittels *high-level*-Funktionen können 2D- und 3D- Visualisierungen und Animationen realisiert werden, während *low-level*-Funktionen für Änderungen der Darstellungsform in den Grafiken und für die Erzeugung von eigenen GUIs verantwortlich sind.
- **Externe Schnittstellen:**
  Das ist die MATLAB-Bibliothek, die die Anbindung von C- und Fortran-Programmen ermöglicht. Diese Schnittstelle kann außerdem für den Aufruf von MATLAB-Routinen verwendet werden.

2.7 Modellierung und Simulation mit Simulink

Als ein Softwarepaket dient Simulink der Analyse, der Modellierung und Simulation dynamischer Systeme, deren Ausgänge sich über die Zeit verändern. Sowohl lineare als auch nichtlineare Systeme können modelliert werden. Diese Modellierung kann bezüglich der Zeit kontinuierlich, diskret oder hybrid stattfinden.

In seiner Struktur beruht Simulink auf einer blockschaltbild-orientierten Darstellungsform. Vorwiegend besitzt Simulink vorprogrammierte Blöcke aus verschiedenen Bereichen. Zusätzlich können selbstprogrammierte Funktionen in Form von *S-functions*[12] hinzugefügt werden.

Um ein dynamisches System in Simulink zu simulieren, sind zwei Schritte nötig:

1. Das System wird in Blockschaltbild-Darstellung mit Hilfe des grafischen *Interface* aufgebaut. Das Modell beschreibt in dieser Konstellation zeitabhängige, mathematische Zusammenhänge zwischen den Eingängen, Ausgängen und Zuständen des Systems.
2. Danach wird das aufgebaute System mit den entsprechenden Einstellungen von Integrationsmethoden und dem Zeitintervall simuliert.

Zunächst wird die Modellierung dynamischer Systeme und später deren Simulation genauer beschrieben.

2.7.1 Modellierung dynamischer Systeme

Um die Modellierung dynamischer Systeme in Simulink zu verstehen, werden zunächst einige Begriffe aus der Simulink-Dokumentation [24] erläutert.

Blockdiagramm

Ein Blockdiagramm in Simulink stellt die grafische Beschreibung des mathematischen Modells dar. Unter dem Begriff mathematisches Modell versteht sich ein Satz von algebraischen, Differential- und Differenzengleichungen.
Ursprünglich stammt diese Beschreibungsform aus den Bereichen Regelungstechnik und Signalverarbeitung. In der grafischen Darstellung bestehen die Diagramme aus Blöcken, die ein dynamisches System bilden, und Linien, die die Signalübertragung im System darstellen. Im Vergleich zum klassischen Blockdiagramm werden die Blöcke in Simulink in zwei Klassen unterteilt: *virtuelle* und *nichtvirtuelle* Blöcke. Während die letzteren die elementaren dynamischen Systeme darstellen, dienen die ersteren der grafischen Übersicht und haben keinen Einfluss auf die Systemgleichungen. Mit den virtuellen Blöcken werden mehrere Blöcke oder Signale zu einem Übersichtsblock oder zu einer Leitung mit mehreren Signalen gebündelt.
Generell werden die Systeme in Simulink mit Hilfe von zeitbasierten Blockdiagrammen beschrieben, die durch folgende Eigenschaften gekennzeichnet sind:

[12] Die *S-functions* können wahlweise in verschiedenen Programmiersprachen, wie z.B. C, ada, fortran, c++ oder Matlab, programmiert werden

- Sie definieren zeitabhängige Zusammenhänge zwischen Signalen und Zustandsvariablen. Die Lösung solcher Blöcke wird durch die Evaluierung der Zusammenhänge über der Zeit innerhalb eines bestimmten Intervalls gegeben. Jede Evaluierung wird in der Simulation als *Zeitschritt* bezeichnet.
- Signale repräsentieren Entitäten, die sich über die Zeit verändern.
- Die oben genannten Zusammenhänge zwischen Signalen und Zustandsvariablen werden als ein Satz von Gleichungen (*Block Methods*) innerhalb eines Blocks definiert. Diese Gleichungen definieren somit die mathematischen Zusammenhänge zwischen Eingangssignalen, Ausgangssignalen und Zustandsvariablen

Aufbau

Elementare Systeme werden für den Aufbau von komplexeren Systemen in Simulink in einer großen, erweiterbaren Bibliothek bereitgestellt. Dieser modulare Aufbau erlaubt eine schnelle und einfache Änderung der Modelle. Einzelne Module können durch andere ersetzt werden. Auch bei der Fehlersuche kann diese Struktur hilfreich sein, da die Module vor dem Einbau ins gesamte System einzeln getestet werden können.

Zustandsvariablen

Dies sind zeitabhängige Variablen, die in jedem Zeitschritt zur Berechnung der Ausgänge eines Modells verwendet werden. Sie werden in jedem Zeitschritt gespeichert, damit sie im nächsten Zeitschritt benutzt werden können. In Simulink unterscheidet man zwei Typen von Zustandsvariablen: kontinuierliche und diskrete Variablen. Dabei sind die diskreten Zustandsvariablen Approximationen von den kontinuierlichen Zuständen in endlichen Zeitintervallen. Die kontinuierlichen Zustände stellen die Grenzwerte dieser Zustände bei unendlich kleinen Zeitintervallen dar.

Modellzustände werden innerhalb von Blöcken implizit definiert. Ein Block, der einige oder alle seiner alten Ausgänge braucht, um die neuen Ausgänge zu berechnen, definiert die Zustände, die innerhalb der Zeitintervalle gespeichert werden müssen. Die Anzahl dieser Zustände wird beim Kompilieren in Simulink berechnet.
Um die kontinuierlichen Zustände des Systems an einem aktuellen Zeitschritt zu berechnen, werden ihre Ableitungen ab dem Simulationsstart integriert. Hierzu gehört natürlich auch die Berechnung der Ableitungen in jedem Zeitpunkt der Simulation. Alle Integrationsvorschriften werden durch Integratoren realisiert. Mit der Reihe ihrer Eingänge stellen sie die

Analogie zu den gewöhnlichen Differenzialgleichungen (ODE) her. Die Integration der Zustände wird in Anlehnung an numerische Methoden (*ODE-Solver*) durchgeführt.
Die Effizienz der Integration der Zustände hängt von der Größe der Zeitintervalle ab. In der Regel führen kleine Zeitintervalle zu physikalisch plausiblen Ergebnissen. ODE-Solver mit variabler Schrittweite können ihre Schrittweite abhängig von der Änderungsgeschwindigkeit der Zustände variieren.

Bei der Berechnung der diskreten Zustände wird der Zusammenhang zwischen der aktuellen Simulationszeit und den unmittelbar zuvor berechneten Werten der Zustände in der *Update*-Funktion beschrieben. Diese Funktionen werden in *diskreten* Blöcken implementiert, die zudem die Abhängigkeit der Zustände von den Systemeingängen definieren.
Im Gegensatz zu kontinuierlichen Zuständen setzen diskrete Zustände Bedingungen für die Schrittweite der Simulation voraus. Der diskrete Solver stellt sicher, dass alle Abtastzeiten aller Modellzustände getroffen werden. Solche Solver werden in zwei Kategorien unterteilt: ein diskreter Solver mit *fester* Schrittweite, die unabhängig von den Zustandsänderungen eingehalten wird, und ein diskreter Solver mit *variabler* Schrittweite, die je nach Änderung von Zustandswerten vergrößert oder verkleinert wird.

Ein System, das sowohl kontinuierliche als auch diskrete Zustände besitzt, wird als hybrides System bezeichnet. Die Wahl der Schrittweite bei der Lösung solcher Systeme muss beide Bedingungen erfüllen: die Genauigkeit der Integration der kontinuierlichen Zustände und das Treffen von Abtastzeiten bei Eintreffen von Ereignissen. Simulink erfüllt diese Bedingungen, indem der nächste Abtastzeitpunkt aus dem diskreten Solver als zusätzliche Bedingung an den kontinuierlichen Solver übergeben wird.

Blockparameter

Die meisten Blöcke in Simulink sind parametrierbare Blöcke. Zum Beispiel hat der Block *Constant* in Simulink einen konstanten Wert. Der Benutzer kann den gewünschten Parameter einstellen, indem er das Dialogfenster des Blocks öffnet und ihn dort eingibt. Man kann diesen Parameter auch in MATLAB eingeben. Er wird dann vor dem Beginn der Simulation auf Gültigkeit geprüft und anschließend verwendet. Zusätzlich kann man solche Parameter während der Simulation ändern. So kann der passende Parameter für das Modell gewählt werden.

Veränderbare Parameter

Dies sind Parameter, die ohne Wiederkompilierung des Modells geändert werden können. Jede Änderung findet am nächsten Zeitschritt statt.
Bei der Eingabe der Option *inline Parameter* werden alle Parameter als unveränderbar angenommen. Dies beschleunigt die Simulationszeit bei komplexen Modellen.

Subsysteme

Ein Blockdiagramm in Simulink besteht aus Schichten, die durch Subsysteme definiert sind. Dabei sind Subsysteme immer als Teil des Blockdiagramms anzusehen und definieren keinesfalls ein neues Blockdiagramm. Sie dienen ausschließlich dem Organisationsaspekt eines Blockdiagramms.
Bei der Ausführung des Modells bereitet Simulink interne Gleichungen vor, die zusammen ausgewertet werden sollen. Die Bearbeitung dieser Systeme wird als „Abflachung“ der Modellhierarchie bezeichnet.
Subsysteme können an ein Ereignis wie z.B. Triggersignal, Funktionsaufruf, Aktion oder Freigabesignal, angeknüpft werden. Deshalb sind sie atomarer Natur und dies bedeutet, dass alle darin beschriebenen Gleichungen als eine Einheit bewertet werden.

Signale

Signale sind zeitlich veränderbare Entitäten, die einen bestimmten abrufbaren Wert in jedem Zeitpunkt einer Simulation haben. Ihnen werden mehrere Attribute zugewiesen wie: Signalname, Datentyp (z.B. 8-Bit, 16-Bit, 32-Bit Integer), numerischer Typ (real oder komplex) und Dimension.

Blockmethoden

Blöcke stellen in erster Linie mehrere Gleichungen eines Systems dar. Diese Gleichungen werden in Simulink als *Block Methodes* bezeichnet. Ihre Evaluierung findet in jeder Simulationsschleife des Blockdiagramms statt.
Blockmethoden werden in der Regel in drei Kategorien unterteilt:

- **Ausgangsfunktion:**
  Hiermit werden die Ausgänge eines Blocks berechnet, dessen Eingänge im aktuellen Zeitpunkt und dessen Zustandsvariablen im vorherigen Zeitschritt vorliegen.

- **Update-Funktion:**
  Hiermit werden die diskreten Zustände eines Blocks im aktuellen Zeitpunkt berechnet, dessen Eingänge im aktuellen Zeitpunkt und dessen Zustandsvariablen im vorherigen Zeitschritt vorliegen.

- **Ableitungsfunktion:**
  Hiermit werden die Ableitungen der kontinuierlichen Zustände eines Blocks im aktuellen Zeitpunkt berechnet. Voraussetzung dafür ist das Vorliegen der Eingänge und der Zustände aus dem vorherigen Zeitschritt.

2.7.2 Simulation dynamischer Systeme in Simulink

Simulation eines dynamischen Systems ist die Bezeichnung eines Prozesses, bei dem die Systemzustände und Ausgänge über einem bestimmten Zeitintervall mittels Informationen aus dem Systemmodell berechnet werden.
Die Realisierung einer Simulation übernimmt die *Simulink Engine*, in dem die in der Abbildung 2.10 beschriebenen Schritte durchgeführt werden.

Kompilierung des Modells

Der Modellkompiler wandelt das Modell in eine ausführbare Form um. Die Aufgaben des Compilers bestehen im Allgemeinen aus:

- Evaluierung der Blockparameter
- Erfassung der Signalattribute wie Name, Datentypen, numerische Typen und Signaldimensionen, die vom Modell nicht explizit festgelegt sind. Danach wird die Anpassung der Signale zu den Blockeingängen überprüft.
- Mit einer sogenannten *Attribut*-Übertragung werden unbestimmte Attribute festgelegt.
- Optimierung der Blockreduktion
- Abflachung der Modellhierarchie, indem virtuelle Subsysteme durch die darin enthaltenen Blöcke ersetzt werden.
- Sortierung der Reihenfolge der Blöcke
- Festlegung der Abtastschrittweite aller Blöcke im Modell, deren Schrittweite noch nicht explizit bestimmt worden ist.

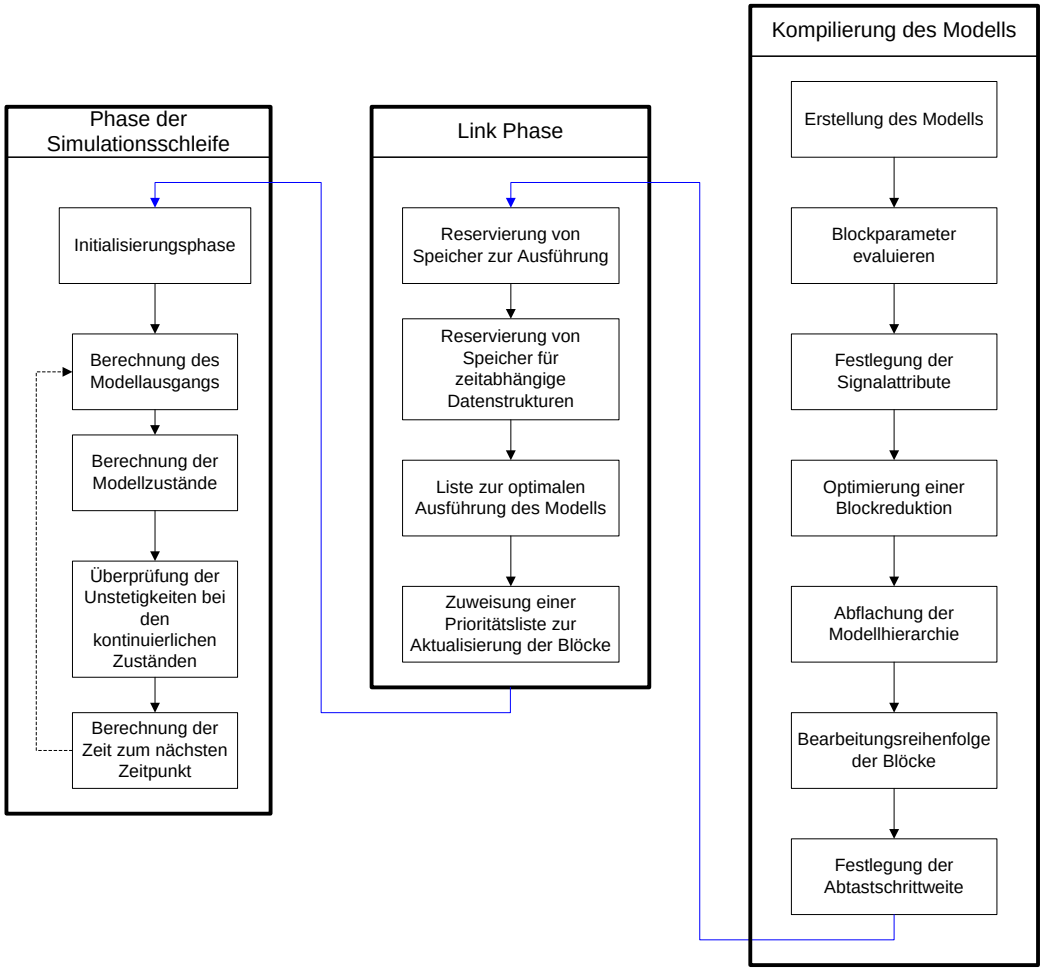


Bild 2.10: Simulationsablauf

Die Link-Phase

In dieser Phase reserviert die *Simulink Engine* den benötigten Speicher zur Ausführung des Blockdiagramms. Weiterhin wird Speicher für zeitabhängige Datenstrukturen aus jedem Block reserviert und initialisiert. Die in Simulink vorprogrammierten Blöcke verfügen über eine Hauptdatenstruktur, genannt *SimBlock*, die *Zeiger*[13] für Blockeingänge, Ausgangspuffer, Zustände und Vektoren speichert.
Außerdem erstellt die *Simulink Engine* die optimale Reihenfolge zur Ausführung des Modells in einer Liste, ausgehend von der in der Kompilierungsphase erstellten Reihenfolge.
Eine weitere Aufgabe dieser Phase ist die Zuweisung einer Prioritätsliste zur Aktualisierung der Blöcke unter Einhaltung der zuvor erstellten Blockreihenfolge.

[13]Ein Zeiger ist eine Variable, die die Adresse einer anderen Variablen enthält. Zeiger werden sehr häufig in der Programmiersprache C benutzt

Die Phase der Simulationsschleife

Mittels der Nutzung der vom Modell zur Verfügung gestellten Information berechnet die *Simulink Engine* die Zustände und die Ausgänge des Modells in kleinen Intervallen, ausgehend vom Beginn der Simulation bis zum Ende. Die aufeinander folgenden Zeitpunkte, bei denen die Berechnung der Zustände und der Ausgänge durchgeführt wird, werden *Zeitschritte* genannt. Die Zeit zwischen zwei Zeitschritten wird als *Schrittweite* bezeichnet. Die letztere hängt von folgenden Faktoren ab: Typ des Gleichungslösers, fundamentale Abtastzeit des Systems und davon, ob die kontinuierlichen Zustände des Systems Unstetigkeiten besitzen.
Diese Phase wird selbst in zwei Subphasen unterteilt: die Initialisierungsphase der Schleife und die Iterationsphase der Schleife. Während die Initialisierungsphase nur einmal am Anfang der Simulation ausgeführt wird, wiederholt sich die Iterationsphase einmal pro Zeitschritt über das gesamte Simulationszeitintervall.
Zunächst werden die Iterationsschritte, die von der *Simulink Engine* in jedem Zeitschritt durchgeführt werden, näher erläutert:

1. **Berechnung der Modellausgänge:**
   Am Anfang eines Schrittes werden die Ausgangsgleichungen des Modells in Simulink initialisiert. Dadurch werden die System-Ausgangsgleichungen aufgerufen, die wiederum die Block-Ausgangsgleichungen nach der in der *Link-Phase* erstellten Reihenfolge aufrufen.
   Die System-Ausgangsgleichungen übergeben an jede Block-Ausgangsgleichung folgende Argumente: einen Zeiger auf die Block-Datenstruktur und auf seine *SimBlock*-Struktur. Die letztere weist Informationen auf, die zur Berechnung der Blockausgänge gebraucht werden, wie die Speicherstellen vom Eingangs- und Ausgangspuffer.

2. **Berechnung der Modellzustände:**
   Hierfür wird ein Gleichungslöser je nach Modellstruktur aufgerufen:

   - Das Modell hat keine Zustände

   - Das Modell besitzt nur diskrete Zustände. In diesem Fall wird der vom Anwender festgelegte diskrete Solver eingesetzt. Er berechnet den richtigen Zeitschritt, um die Abtastzeit des Modells zu treffen. Danach wird die *Update-Method*[14] des Modells aufgerufen. Somit wird die U*pdate-Method* des Systems und dann des Blocks erneut aufgerufen. Die Reihenfolge des Aufrufs wird von der in der Link-Phase erstellten Liste übernommen.

[14] Gleichungen zur Aktualisierung des Modells

- Das Modell hat nur kontinuierliche Zustände. In diesem Fall wird der vom Modell festgelegte kontinuierliche Solver eingesetzt. Je nach Solver werden reihum
  - die Ableitungsmethoden des Modells aufgerufen,
  - innerhalb eines Major-Zeitschritts, werden in Minor-Zeitschritten sukzessiv die Modellableitungen und Modellausgänge berechnet.
- Das Modell hat sowohl kontinuierliche als auch diskrete Zustände.

3. Gegebenenfalls findet eine Überprüfung von Unstetigkeiten bei den kontinuierlichen Zuständen statt. Hierfür wird die ZCD[15]-Technik benutzt. Diese Technik wird im Rahmen dieser Arbeit in Kapitel 5 näher erläutert.
4. Berechnung der Zeit zum nächsten Zeitschritt.

Diese Schritte werden solange wiederholt, bis die Simulationsendzeit erreicht wird.

Numerische Gleichungslöser

Der Prozess, in dem die sukzessiven Zustände eines Systems anhand des Modells berechnet werden, wird als *Lösung des Modells* bezeichnet. Da keine einzelne Methode ausreicht, um alle Systeme erfolgreich zu lösen, stellt Simulink eine ganze Reihe von Programmen zur Verfügung, bekannt als *Solver*, die diese Aufgabe je nach Modellart übernehmen.
Simulink Solver werden in zwei Kategorien unterteilt: Solver mit fester Schrittweite und Solver mit variabler Schrittweite.
Solver mit fester Schrittweite lösen das Modell in einem bestimmten Zeitintervall. Die Größe des Intervalls wird als Schrittweite bezeichnet, die entweder vom Anwender oder vom Solver selbst bestimmt wird. In der Regel führt eine kleine Schrittweite zu genaueren Ergebnissen, jedoch nimmt so die Rechenzeit der Simulation zu.
Solver mit variabler Schrittweite reduzieren die Schrittweite, um eine große Genauigkeit bei schnell veränderlichen Modellzuständen zu erzielen. Um unnötige Schritte zu sparen, wird bei langsam veränderlichen Modellzuständen die Schrittweite vergrößert. Obwohl die Berechnung der Schrittweite einen zusätzlichen Rechenaufwand mit sich bringt, wird die gesamte Anzahl von Rechenschritten so reduziert. Somit wird ein gewisses Maß an Genauigkeit für ein Modell mit schnell veränderlichen Zuständen oder stückweise kontinuierlichen Zuständen eingehalten. Um den Toleranzrahmen einzuhalten, werden häufig Gleichungslöser mit Prädiktor-Korrektor-Verfahren (siehe Kapitel3) eingesetzt.
Weiterhin kann man die Solver in Simulink in kontinuierlich und diskret unterteilen.

[15]Zero Crossing Detection

Kontinuierliche Solver führen numerische Integrationen durch, um die kontinuierlichen Zustände eines Modells in einem aktuellen Zeitpunkt aus den Zuständen des vorherigen Zeitpunktes und der Zustandsableitungen zu berechnen. Um die diskreten Zustände in jedem Zeitschritt zu berechnen, nutzen kontinuierliche Solver die jeweiligen Modellblöcke. Simulink verfügt über eine große Anzahl an kontinuierlichen Solvern mit fester oder variabler Schrittweite, die spezifische Lösungen für gewöhnliche Differenzialgleichungen implementieren.
Diskrete Solver werden ausschließlich für die Berechnung des nächsten Simulationszeitschritts in einem *rein diskreten* Modell benutzt. Dazu benutzen sie die in Modellblöcken aktualisierten diskreten Zustände.
Da ein diskreter Solver keine kontinuierlichen Zustände berechnen kann, werden Modelle mit sowohl kontinuierlichen als auch diskreten Zuständen ausschließlich mit kontinuierlichen Solver gelöst.
Einige kontinuierliche Solver unterteilen das Zeitintervall in *Minor*-Zeitschritte und *Major*-Zeitschritte. Dabei sind *Minor*-Zeitschritte Unterteilungen von *Major*-Zeitschritten. Ergebnisse werden erst am Ende des *Major*-Zeitschritts erzeugt.

Diskontinuität

In einem dynamischen System können Diskontinuitäten in verschiedenen Formen vorkommen:

- Diskontinuitäten in den Stimuli-Funktionen: In den betroffenen Punkten sind diese Funktionen entweder unstetig oder nicht differenzierbar.

- Diskontinuität infolge von Veränderungen der systembeschreibenden Gleichungen: diese Änderungen können sich entweder auf die Struktur der Gleichung (Gleichung 2.1) oder auf die Parameter (Gleichung 2.2) beziehen.

$$\underline{\dot{x}} = f(\underline{x}, t) \quad \Rightarrow \quad \underline{\dot{x}} = g(\underline{x}, t) \tag{2.1}$$

$$x = f(x, t) = ax(t) + b \quad \Rightarrow \quad \dot{x} = \acute{f}(x, t) = \acute{a}x(t) + \acute{b} \tag{2.2}$$

Es gibt eine Simulink-Funktion, mit deren Hilfe Unstetigkeiten in den Zustandsvariablen des Systems oder deren Ableitungen in jedem Zeitschritt erkannt werden können. Diese Technik wird in Kapitel 5 näher erläutert.

Algebraische Schleifen

Einige Blöcke in Simulink haben direkte Verbindungen zwischen Eingang und Ausgang. Daher können die Ausgänge dieser Blöcke erst mit dem Vorhandensein der Werte der Eingangssignale berechnet werden.

Tabelle 2.3: Simulink-Blöcke mit direktem Durchgriff

Math Function
Gain
Integrator
Product
State-Space
Sum
Transfer Function
Zero-Pole

Eine algebraische Schleife tritt dann auf, wenn ein Eingangsport eines Blocks mit direktem Durchgriff durch einen Ausgangsport desselben Blocks mitangesteuert wird. Abbildung 2.11 ist ein Beispiel solcher algebraischen Schleifen.

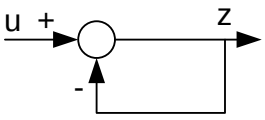


Bild 2.11: Beispiel einer algebraischen Schleife

Mathematisch gesehen zwingt die Schleife den algebraischen Zustand z gleich dem Inputsignal u minus z zu sein. Die Lösung dieser einfachen Schleife ist $z = u/2$; aber die meisten algebraischen Schleifen können nicht so einfach gelöst werden. In den meisten der Fälle kann der Block *Algebraic Constraint* eine Hilfe schaffen. Dazu wird ein algebraischer Schleifenvektor mit mehreren algebraischen Zustandsvariablen $z1$, $z2$, usw. erstellt, deren Initialwerte im oben erwähnten Block geschätzt werden. Dafür wird das Eingangssignal $F(z)$ auf Null gesetzt und der Ausgang z so eingestellt, dass Null im Eingang produziert wird. Um die Effizienz des Verfahrens zu steigern, können Schätzwerte vom Anwender angegeben werden.

2.7.3 Modellierung und Simulation diskreter Systeme

Mit Simulink können diskrete Systeme simuliert werden, einschließlich Systeme mit verschiedenen Abtastfrequenzen und hybride Systeme, die sowohl kontinuierliche als auch diskrete Komponenten besitzen. Diese Fähigkeiten verdankt Simulink zwei Eigenschaften:

- **Das Blockparameter *SampleTime*:**
  Einige Simulink-Blöcke haben einen Parameter, genannt *SampleTime*, mit denen sich die Abtastzeiten des Systems bestimmen lassen. Dieser Parameter kann entweder implizit wie bei kontinuierlichen Blöcken oder explizit vom Anwender festgelegt werden.

- **Vererbung von Abtastzeiten:**
  Die meisten *Standardblöcke* in Simulink können ihre Abtastzeiten von den mit ihren Eingängen verbundenen Blöcken erben.

Mit den oben genannten Eigenschaften kann die Abtastzeit eines Blocks entweder direkt durch einen Parameter oder indirekt durch Vererbung aus anderen Blöcken festgelegt werden. Dies erlaubt eine Implementierung von Systemen mit mehreren Abtastzeiten oder hybride Systeme mit kontinuierlichen und diskreten Komponenten.

Festlegung der Abtastzeit

Der *SampleTime*-Parameter in Simulink kann entweder allein als Skalar oder in Form eines Vektors $[T_s, T_0]$ eingegeben werden, dessen erstes Element die Abtastzeiten und dessen zweites Element ein Offset ist.

Kompilierte Abtastzeit Während der Kompilierungsphase bestimmt Simulink die Abtastrate eines Blocks anhand seines *SampleTime*-Parameters durch die Vererbung der Abtastzeit aus anderen Blöcken oder durch den Typ des Blocks. Diese kompilierte Abtastzeit bestimmt die Abtastrate des Systems für die weitere Verarbeitung in Simulink.

Rein diskrete Systeme

Um ein System zu simulieren, das nur diskrete Komponenten beinhaltet, wird ein diskreter Solver ausgewählt. Somit werden Ausgangspunkte nur an den Abtastpunkten generiert. Folglich weisen die Ergebnisse keine Unterschiede auf.

Bestimmung der Schrittweite für diskrete Systeme

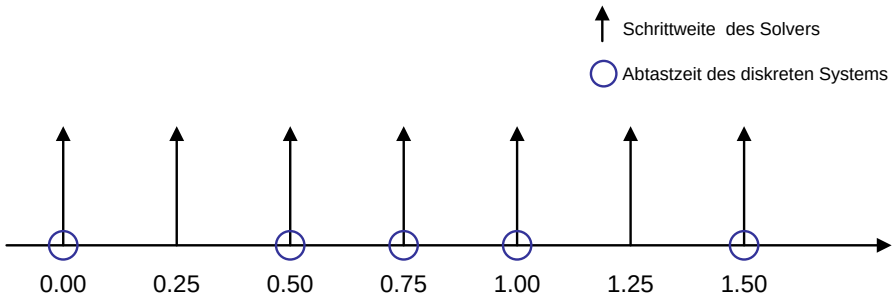
Um ein diskretes System zu simulieren, muss der Simulator einen Simulationsschritt in jedem Abtastzeitpunkt durchführen. Dazu wird ein Simulationsschritt in jedem ganzzahligen Vielfachen der kürzesten Abtastzeiten im System errechnet. ($t_{Schritt:k} = k \cdot T_{Abtast:0}$; $n \in \mathcal{N}$ und $T_{Abtast:0}$: die kürzeste Abtastperiode im System)
Um die Abtastzeitpunkte des zu simulierenden digitalen Systems zu treffen, legt Simulink die Simulationsschrittweite abhängig von der fundamentalen Schrittweite des Systems und von dem Typen des Solvers fest.
Die *fundamentale Abtastzeit* eines diskreten Systems ist der größte ganzzahlige Teiler der aktuellen Abtastzeit des Systems. Ein System, das z.B. die Abtastzeiten 0.25 und 0.5 Sekunden besitzt, hat die fundamentale Abtastzeit von 0.25 Sekunde. Angenommen die Abtastzeiten sind 0.5 und 0.75 Sekunden, dann ist die fundamentale Abtastzeit auch 0.25 Sekunden.
Ein diskretes System kann in Simulink entweder mit einem diskreten Solver mit fester Schrittweite oder einem diskreten Solver mit variabler Schrittweite simuliert werden. Ein Solver mit fester Schrittweite legt die Schrittweite gleich der fundamentalen Abtastzeit des diskreten Systems fest. Ein Solver mit variabler Schrittweite ändert dagegen seine Schrittweite, um die aktuellen Abtastzeitpunkte genau zu treffen. Die Abbildungen 2.12(a) und 2.12(b) zeigen den Unterschied zwischen den beiden Solvern.
Die beiden Diagramme 2.12(a) und 2.12(b) zeigen, dass der Solver mit variabler Schrittweite weniger Simulationsschritte als der Solver mit fester Schrittweite aufweist, wenn die fundamentale Abtastzeit kleiner als irgend eine aktuelle Abtastzeit des zu simulierenden Systems ist. Auf der anderen Seite verbraucht der Solver mit fester Schrittweite weniger Speicher bei der Implementierung und ist im Allgemeinen schneller, wenn eine Abtastzeit im System gleich der fundamentalen Abtastzeit ist.

Konstante Abtastzeit

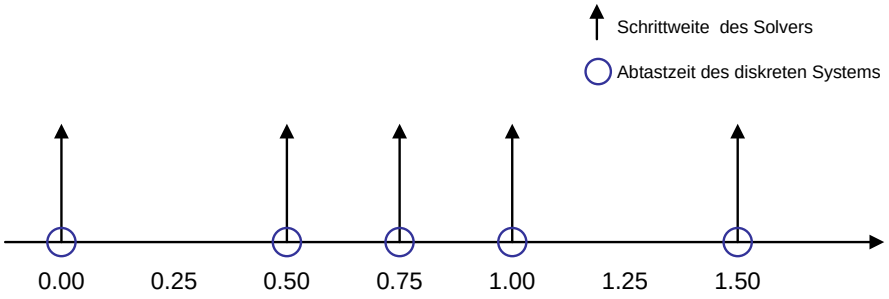
Ein Block, dessen Ausgang sich während der gesamten Simulation nicht ändert, wird als *Block mit konstanter Abtastzeit* genannt. Um diese Eigenschaft zu erfüllen, müssen Blöcke zwei Bedingungen erfüllen:

- All ihre Parameter sind nicht veränderbar; entweder weil sie diese Eigenschaft vererbt haben oder weil sie *inlined*[16] sind.

[16]Das ist eine Option in Simulink, mit der die Abstimmbarkeit der Parameter von Blöcken abgeschaltet wird. Somit können Blöcke, deren Ausgänge nur von diesen Parametern abhängen, außerhalb der Simulationsschleife errechnet werden und so beschleunigt sich die Simulation und die Ausführung des Codes.



(a) Solver mit fester Schrittweite



(b) Solver mit variabler Schrittweite

Bild 2.12: Schrittweitendiagramme von verschiedenen *Solvern*

- Die Abtastzeit des Systems wurde als unendlich definiert oder sie wurde aus einem anderen Block mit konstanter Abtastzeit vererbt.

Zu Beginn der Simulation aktualisiert Simulink beispielsweise das Modell und bestimmt welche Blöcke eine konstante Abtastzeit besitzen. Daraufhin wird der Initialwert des Ausgangsports berechnet. Sollte der Ausgang dieser Blöcke mit konstanter Abtastzeit verwendet werden, dann greift Simulink auf die zuvor berechneten Initialwerte zurück. Somit wird unnötige Rechenzeit vermieden.

Kontinuierlich/diskrete Systeme

Gemischte Systeme, kontinuierlich und diskret, bestehen aus abgetasteten und kontinuierlichen Blöcken. Zur Simulation dieser Systeme können alle Integrationsmethoden verwendet werden, vorzugsweise Gleichungslöser mit variabler Schrittweite, die meistens effiziente und genaue Ergebnisse liefern.

3 Stand der Technik

3.1 Einführung

Mit der steigenden Komplexität technischer Systeme nimmt die Bedeutung der Konzeptionsphase immer mehr zu. Die modernen technischen Systeme verfügen über immer mehr Komponenten, die aus verschiedenen Ingenieursdisziplinen stammen. Für jede Disziplin existieren auf dem Markt zahlreiche Werkzeuge mit deren Hilfe der Entwurf und die Analyse jedes bestimmten Teilsystem virtuell durchgeführt werden kann. Angesichts der großen Überschneidungen dieser Bereiche miteinander, wächst die Nachfrage nach geeigneten *Interface-Strukturen* mit deren Hilfe sich eine umfangreiche Simulationsumgebung aufbauen lässt.
Vor allem im Bereich der Automatisierungstechnik, wo mechanische Mehrkörpersysteme mit Regeleinrichtungen verknüpft werden, steht man seit einem Jahrzehnt vor diesem Dilemma, ob es rentabler ist, ein einziges optimales Werkzeug für alle Simualtionsanforderungen aus verschiedenen Bereichen zu finden bzw. zu entwickeln oder ob die Entwicklung robuster *Interface-Strukturen* innerhalb der vorhandenen Werkzeuge reicht, um Werkzeugverbünde zu erhalten, welche die Anforderung nach einer Simulation des gesamten Systems erfüllen? Langsam kommt man zu der Überzeugung, dass die zweite Alternative viel mehr Vorteile mit sich bringt als die erste. Dies liegt daran dass die Entwicklung geeigneter Schnittstellen weniger Aufwand erfordert. Außerdem können Werkzeughersteller weiterhin an der Verbesserung ihres Produktes arbeiten, ohne andere Disziplinen miteinzubeziehen. Es gibt jedoch bis jetzt keine standardisierte Vorgehensweise, sondern es wird je nach Problemstellung und Randbedingungen immer wieder neu entschieden.

In der Fahrzeug-Industrie wird zunehmend nach einem Standardansatz gesucht, mit dem die mechanischen Mehrkörpersysteme zusammen mit der Elektronik im Fahrzeug analysiert und simuliert werden können. Dieser Verbund aus der Realität wird mit einem Werkzeugverbund in der virtuellen Welt nachgebildet. Es geht also um eine Kombination aus Mehrkörpermodellierungssoftware und regelungstechnischer Software zur rechnergestützten Simulation.

Die Grundlage zur Kopplung der Simulationsprogramme untereinander bildet der Datenaustausch zwischen den einzelnen Werkzeugen. Es werden unter anderem Datenstrukturen und Parametersätze ausgetauscht. Definieren diese Daten- und Parametersätze ein Modell, so spricht man von einem *Modellaustausch*.

Im Allgemeinen besteht ein Werkzeug aus:

- Werkzeugumgebung bestehend aus Modell-Editor und eine Ergebnis-Visualisierungseinheit
- Gleichungslöser (Solver)
- Modellen

Es kann zwischen zwei Hauptansätzen zur Kopplung der Programme entschieden werden: Integration auf Gleichungslöser- und auf Modellebene. Die detaillierte Vorgehensweise wird aber je nach Problemstellung und Randbedingung festgelegt.

Neben den oben genannten Ansätzen existieren auf dem Markt verschiedene Produkte zur softwareunabhängigen Kopplung, die den Fachgebieten Datentechnik und reiner Kommunikationstechnik zugeordnet werden können. Sie werden im nächsten Abschnitt (3.2) kurz zusammengefasst.

In der vorliegenden Arbeit beschränkt sich der Autor auf die Kopplung zwischen dem Mehrkörpersimulationsprogramm ADAMS und der Regelungstechniksoftware Matlab/Simulink, die im vorherigen Kapitel vorgestellt worden sind. In diesem Kapitel werden die beiden Kopplungsmethoden in 3.3 vorgestellt. Anschließend wird im Unterkapitel 3.4 die DLL (Dynamic Link Library) näher beschrieben, die als Datenaustauschformat bei der Modellkopplung in der vorliegenden Arbeit verwendet wird. Danach werden im Unterkapitel 3.5 die beiden Solver sowohl in ADAMS als auch in Simulink erläutert, die als Basis zur Kopplung auf Solverebene dienen.

3.2 Elektronischer Datenaustausch

In der Fachliteratur [68] wird der *elektronische Datenaustausch* so definiert:

Definition Das Wort *elektronischer Datenaustausch* ist ein Sammelbegriff aller elektronischen Verfahren zum asynchronen und vollautomatischen Versand von strukturierten Nachrichten zwischen Anwendungssystemen unterschiedlicher Institutionen.

Ein Beispiel dieser Anwendungssysteme sind Simulationswerkzeuge aus verschiedenen Ingenieurdisziplinen, die in einer gemeinsamen Simulationsumgebung integriert werden. Es wurden bis heute sehr viele Ansätze zur Entwicklung einer standardisierten Schnittstelle vorgeschlagen, die unabhängig von den Tools alle Formen des elektronischen Datenaustausches unterstützt.
Um einen Überblick über die verschiedenen Produkte zur werkzeugunabhängigen Kopplung von Simulationsumgebungen zu geben, werden in Anlehnung an die Studie von Arenz und Schnieder [18] die wichtigsten Ansätze in diesem Gebiet hier aufgelistet.

3.2.1 CORBA

(**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) ist eine von der OMG (Object Management Group) standardisierte, plattformunabhängige Middelware Architektur. Sie basiert auf einem objektorientierten Kommunikationsinterface mit *Server-Client-Struktur*. Es wird die IDL (Interface Definition Language) zur Definition des Interface verwendet, welches die einzelnen Objekte kapselt und die Interfaceformen standardisiert.

Vorteilhaft bei diesem Ansatz ist die Parallelisierung der Aufgaben. Dagegen fällt auf, dass die Interfacestrukturen vom Benutzer selbst programmiert und implementiert werden müssen. Dieser Vorgang ist zeitaufwendig und kompliziert. Es werden außerdem fundierte Kenntnisse der zu koppelnden Programme und Modelle vorausgesetzt, was meistens nicht der Fall ist.

Insbesondere für Programme wie ADAMS, mit unbekannten Programmierstrukturen, kann es sehr schwierig sein, ein CORBA-Interface mit anderen Programmen zu entwickeln, das alle Anforderungen einer gekoppelten Simulation erfüllt.

3.2.2 Webtechnologie

Ähnlich wie bei CORBA basiert der Informationsaustausch einer Webtechnologie auf einer Client-Server-Architektur. Vorwiegend werden hierfür TCP/IPs (Transmission Control Protocol/Internet Protocol) und HTTP (Hypertext Transfer Protocol) als Kommunikationsprotokolle verwendet.

3.2.3 STEP

(**ST**andard for the **E**xchange of **P**roduct Model Data) ist ein internationaler Standard zum Austausch von Produktdaten zwischen verschiedenen Software-Produkten wie CAD (Computer Aided Design), CAM (Computer Aided Manufacturing), DMU (Digital Mock-Up) und PDM (Product Daten Management). Es basiert auf einem neutralen Datenformat, das von jedem Teilnehmer des Werkzeugverbundes genutzt, gelesen und umgeschrieben werden kann. Innerhalb von STEP können Produktdateninformationen des gesamten Lebenszyklus abgebildet werden. Um die Anwendungsbereiche von STEP zu vergrößern, werden sogenannte Applikationsprotokolle für die verschiedenen Einsatzgebiete zur Verfügung gestellt.

3.3 Werkzeugabhängige Kopplungsmethoden

3.3.1 Motivation

Komplexe mechatronische Systeme können entweder mit einem Simulationswerkzeug oder mit mehreren Werkzeugen simuliert werden. Bei der ersten Lösung wird ein Werkzeug verwendet, dessen Schwerpunkt in dem Bereich einer Disziplin liegt, während die Komponenten aus anderen Disziplinen mit vereinfachten Modellen berücksichtigt werden. Auf diese Weise wird auf Kosten der Modellierungstiefe am Modellierungsaufwand gespart. Bei der zweiten

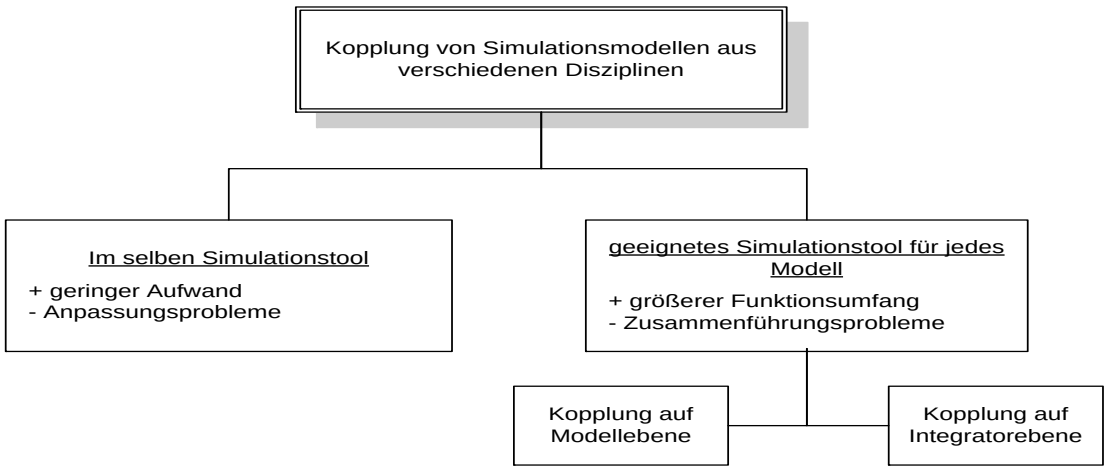


Bild 3.1: Simulation von Systemen bestehend aus Komponenten verschiedener Disziplinen

Variante verwendet man für jeden Disziplin ein Werkzeug. Dadurch entstehen Modelle mit

hohem Detaillierungsgrad, aber der Anwender wird mit dem Kopplungsproblem konfrontiert. Die Abbildung 3.1 gibt eine Übersicht über diese beiden Kopplungsmethoden. Ziel der *werkzeugabhängigen Kopplung* von Simlationstools ist die Entwicklung einer für den Anwender leicht bedienbaren Schnittstelle. Wir werden in der vorliegenden Arbeit auf bestehenden Simulationsprogrammen und deren Schnittstellen aufbauen. Dadurch bekommt man eine problemorientierte Schnittstelle. Sie wird in ihrem Aufbau an die technischen Anforderungen des Systems und an die Bedienungsanforderungen des Anwenders angepasst. Somit wird der Aufwand einer umfassenden Gesamtlösung vermieden und man kann sich auf die bereits existierende Interfacestruktur konzentrieren. Eine umfassende Untersuchung gibt dann Aufschluss darüber, wie diese Schnittstelle erweitert werden soll, um die Simulationszuverlässigkeit zu gewährleisten. Die Tabelle 3.1 listet die Vor-und Nachteile der beiden Verfahren auf: Weil ein Simulationswerkzeug auf einer Beschreibungsmethode mit einem bestimmten

Tabelle 3.1: Simulation mit einem Werkzeug vs. Simulation mit mehreren Werkzeugen

	Simulation mit nur einem Werkzeug	**Simulation mit mehreren Werkzeuge**
Vorteile	geringerer Modellierungsaufwand schnellere Prozesse	größerer Funktionsumfang größerer Detaillierungsgrad
Nachteile	kleinerer Funktionsumfang grobe Modellierung	Probleme bei der Kopplung

Detaillierungsgrad basiert, gleicht eine Kopplung von mehreren Programmen in der Regel der Einebnung ihrer Modelle auf einem gemeinsamen Abstraktionslevel oder dem Zugriff auf eine Datenschnittstelle, die von beiden Programmen bedient wird (siehe Abbildung 3.2).

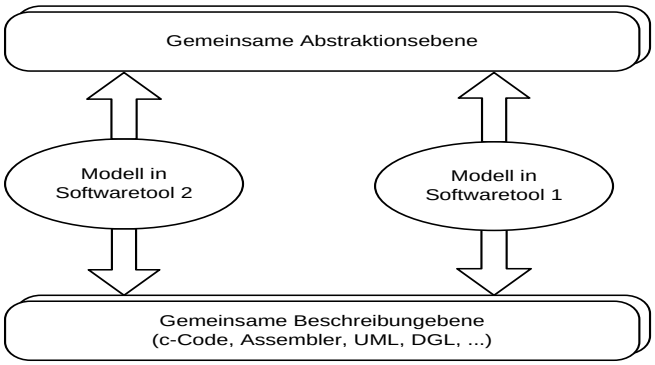


Bild 3.2: Prinzip der Kopplung von zwei Simulationstools

Generell besteht ein Simulationsprogramm aus einem Integrator und einem Modelldatensatz, daher beruht die Kopplung auf einem der beiden Kerne. Wir werden zunächst von Kopplungen auf *Modellebene* und dann von Kopplungen auf *Integratorebene* sprechen. Die Abbildung 3.3 gibt eine Übersicht über diese beiden Kopplungsformen.

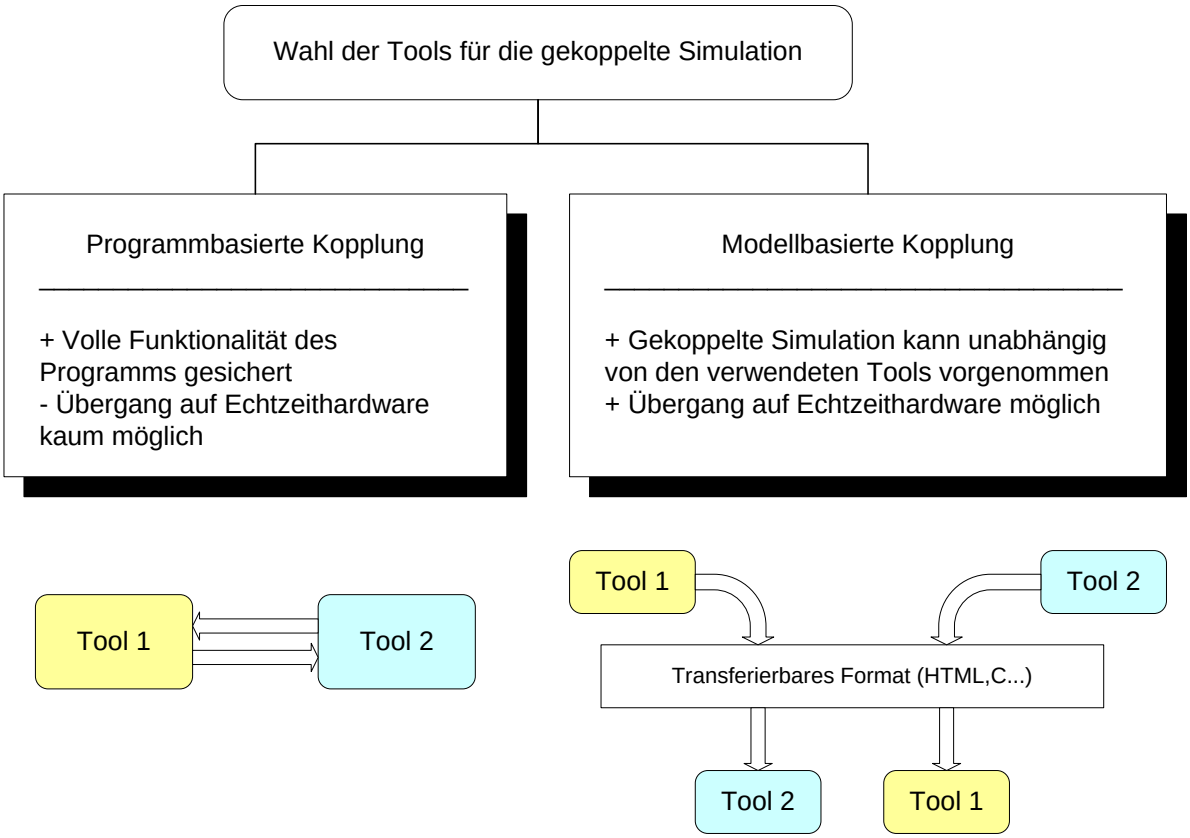


Bild 3.3: Kopplungsformen von Simulationsprogramme

3.3.2 Kopplung auf Modellebene

In der vorliegenden Arbeit wird der Begriff Kopplung auf Modellebene verwendet für Kopplungen bei denen Modelle aus einem Werkzeug ins andere exportiert werden. Die Simulationen werden dann im Zielwerkzeug mit der Hilfe des eigenen Gleichungslösers durchgeführt.

Dieses Verfahren verwendet man auch für die Kopplung der MKS-Software ADAMS und der regelungstechnischen Software MATLAB/Simulink unter dem Namen *integrierte Simulation*. Die Abbildung 3.4 zeigt die Analogie zwischen der Simulationsumgebung in der *integrierten Simulationskonstellation* und dem realen Fahrzeugsystem mit den entsprechenden Fahrerassistenzsystemen. Da die zu integrierenden Modelle vor der Simulation in die Zielsoftware

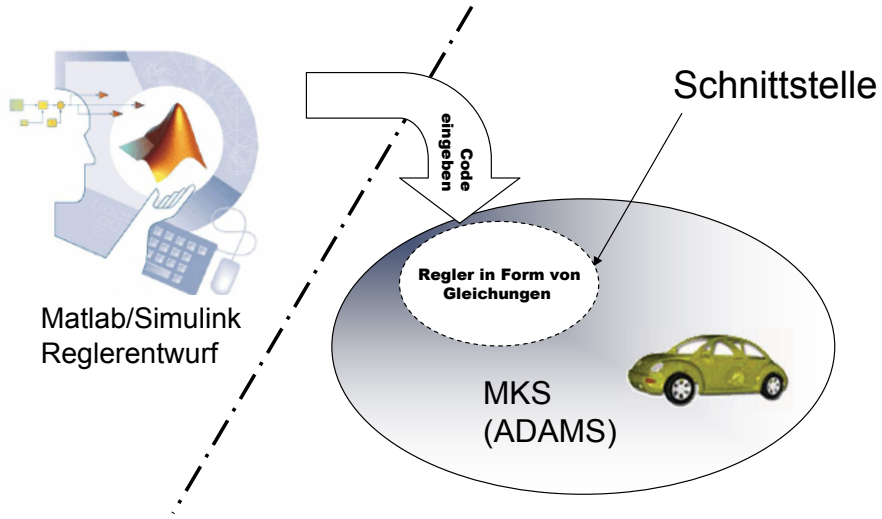


Bild 3.4: Kopplung auf Modellebene: integrierte Simulation

exportiert werden, entsteht keine Kommunikation zwischen den beiden Programmen während der Simulation. Daher spricht man hier von einer *offline Simulation*.

3.3.3 Kopplung auf Integratorebene

In dieser Einstellung kommunizieren beide Simulationsprogramme untereinander kontinuierlich. In jedem Integrationsschritt werden Daten zwischen beiden Gleichungslösern ausgetauscht. In der Literatur hat der Begriff *Co-Simulation* für dieses Kopplungsverfahren eine breite Verwendung.
Oft besteht der Softwareverbund aus einem *Master*, von dem die Simulation gesteuert wird, und einem *Client*, der die Systemdaten in jedem Integrationsschritt für den Master zur Verfügung stellt. Da die Simulationsdaten zwischen beiden Tools in jedem Simulationsschritt ausgetauscht werden, wird hier die Bezeichnung *Online Kopplung* sehr oft benutzt.

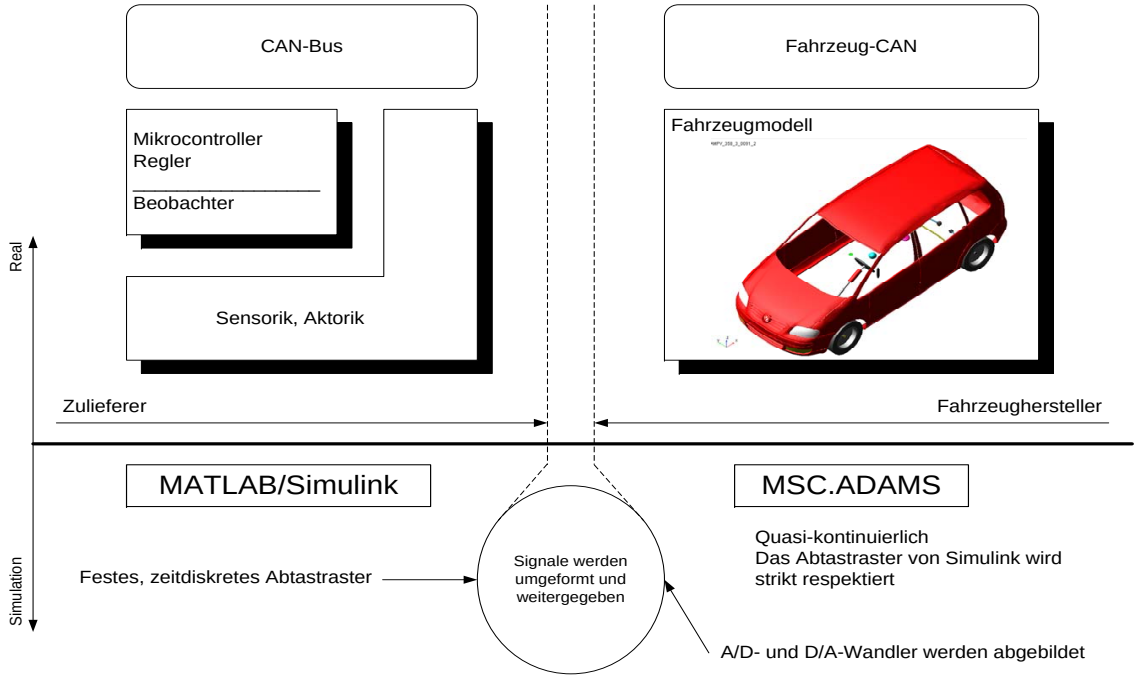


Bild 3.5: Kopplung auf Integratorebene: Co-Simulation

3.4 Dynamic Link Libraries (DLL)

DLL sind Laufzeitmodule, die ursprünglich zur Auslagerung von C++-Klassen gedacht waren. *Clientprogramme* können die DLLs zur Laufzeit der Simulation schnell laden und linken. Diese Eigenschaft wird bei der Offline-Kopplung von Simulationsprogrammen verwendet. Somit werden „*Black-Box*“ Modelle in Form einer DLL in einer Software entwickelt und zur anderen exportiert. Zunächst wird in diesem Unterkapitel die DLL genauer beschrieben. Danach wird der Mechanismus der Importe und Exporte und anschließend die Festlegung des Eintrittspunktes in die DLL vorgestellt. Zum Schluss werden die Suchpfade der DLLs für das Clientprogramm aufgelistet.

3.4.1 Grundlagen

Eine DLL ist im Wesentlichen eine Datei auf einem Datenträger, die normalerweise die Namenserweiterung DLL trägt, globale Daten, kompilierte Funktionen und Ressourcen enthält und in einen Prozess integriert wird. Hierbei ist ein Prozess eine ausgeführte Instanz eines Programms, das als EXE-Datei auf dem Datenträger vorliegt.
Eine DLL wird für eine bevorzugte Basisadresse kompiliert und wenn kein Konflikt mit an-

deren DLLs auftritt, wird die Datei an dieser virtuellen Adresse in den Prozess eingebunden. Das Clientprogramm kann alle Funktionen sowie globale Variablen importieren, die von der DLL exportiert worden sind.

3.4.2 Importe und Exporte

Die von der DLL exportierten Funktionen werden in einer Tabelle aufgelistet. Jede Funktion besitzt einen symbolischen Namen und optional einen ganzzahligen Wert, die Ordinalzahl, mit der sie eindeutig identifiziert werden kann. Anhand einer der beiden *Identifier* erkennt das Clientprogramm die enthaltenen Funktionen beim Laden der DLL. Die Funktionstabelle enthält außerdem die Adressen der Funktionen innerhalb der DLL. Beim dynamischen Linken wird eine zusätzliche Tabelle erstellt, die den Aufrufen aus dem Clientprogramm die entsprechenden Funktionsadressen der DLL zuordnet. Dies führt dazu, dass Clientprogramme erst dann neu erstellt werden müssen, wenn die Funktionsnamen oder Parameterlisten in einer der geladenen DLLs verändert worden sind. Nachdem die DLL geladen wurde, findet während einer Simulation ein Aufruf der DLL in jedem Simulationsschritt statt.
Beim Aufrufen einer DLL muss man darauf achten, dass die DLL Funktionen aus anderen DLLs aufgerufen werden können. Beim dynamischen Linken wird die Wechselwirkung zwischen *Importen* und *Exporten* berücksichtigt.
Im Quelltext der DLL wird eine exportierte Funktion so deklariert:
__declspec(dllexport) int Funktionsname(int n);
Aus dem Client-Programm sieht der Import so aus:
__declspec(dllimport) int Funktionsname(int n);
In den oben eingegebenen Deklarationen wird der Name der Funktion durch einen *ergänzten* Namen ersetzt, der in der MAP-Datei eines Projekts aufgelistet wird. Wenn aber der ursprüngliche Name der Funktion für Deklarationen in anderen Modulen verwendet werden soll, dann müssen die *Importe* und *Exporte* wie folgt eingegeben werden:
extern “C“ __declspec(dllexport) int Funktionsname(int n);
extern “C“ __declspec(dllimport) int Funktionsname(int n);

Um den Einbindungsprozess der DLL zu vervollständigen, muss neben den Importdeklarationen die Importdatei beim Linker[1] angemeldet werden und mindestens eine der importierten Funktionen im Ausführungspfad des Programms tatsächlich aufrufen.

[1] Unter einem Linker versteht man ein Programm, das einzelne Programmmodule zu einem ausführbaren Programm zusammenstellt [68].

3.4.3 Eintrittspunkt in die DLL

Den Eintrittspunkt einer DLL legt der Linker mit der Funktion `_DllMainCRTStartup` fest. Diese Funktion wird zuerst beim Laden einer DLL aufgerufen. Sie ruft wiederum die Konstruktoren der globalen Objekte und anschließend die globale Funktion `DllMain` auf.

3.4.4 Suchpfade

Bei einer expliziten Einbindung der DLL über die Funktion `LoadLibrary` kann der vollständige Pfadname der DLL angegeben werden. Sollte aber die DLL implizit eingebunden werden oder der Pfadname fehlen, wird die DLL in den folgenden Verzeichnissen nach dieser Reihenfolge gesucht:

1. im Verzeichnis, das die EXE-Datei enthält,
2. im aktuellen Verzeichnis des Prozesses,
3. im Windows-Systemverzeichnis,
4. im Windows-Verzeichnis,
5. in den Verzeichnissen, die in der Umgebungsvariable *Path* angegeben sind.

Während einer integrierten Simulation können Modellgleichungen in der DLL zur Verfügung stehen. Sie wird vom Gleichungslöser in jedem Simulationsschritt aufgerufen. Die DLL liefert dann je nach Modellbeschreibung entweder Vorschriften wie der Gleichungslöser die Gleichungen lösen soll oder direkt die Lösung der Gleichungen in diesem Simulationsschritt. Da die Lösung der Gleichungen stark vom Lösungsverfahren abhängt, werden zunächst die in ADAMS und Simulink meist verwendeten Gleichungslöser in einem Überblick vorgestellt.

3.5 Überblick über die Gleichungslöser

Simulationsprogramme bestehen grundsätzlich aus Beschreibungsmodellen und Gleichungslösern. Die Beschreibungsmodelle sind meistens Sätze von Gleichungen, die während der Simulationen in jedem Integrationsschritt gelöst werden müssen. Um die Kopplung zwischen den Simulationsprogrammen ADAMS und MATLAB/Simulink besser zu verstehen ist also eine Beschreibung der Gleichungen und der Gleichungslösung in beiden Programmen notwendig.

Um die Simulationen durchzuführen, werden gewöhnliche Differentialgleichungen des Systems aufgestellt. Anhand des Modellaufbaus wird neben den Bewegungsgleichungen das Anfangswertproblem aufgestellt. Da das implizite Lösungsverfahren meistens unbekannt ist, wird die Lösung aus den Systemgleichungen numerisch errechnet.

Im folgenden Unterkapitel werden in Anlehnung an [66], [24] und andere Literatur aus der numerischen Mathematik die wesentlichen Integrationsmethoden beschrieben, die in ADAMS und in Matlab/Simulink Verwendung finden.

3.5.1 Anfangswertproblem eines MKS

Vorausgesetzt wird ein Differentialgleichungssystem mit den expliziten Gleichungen:

$$\dot{z} = f(z,t) \quad ; \qquad z_0 = z(0) \qquad \text{Anfangszustand}$$

Dabei ist:

- $z = z(t)$ der Zustand des Systems am Simulationszeitpunkt t
- $t \in [0,T]$ das Integrationszeitintervall

Für eine Mehrkörpersimulation gilt:

$$z = \begin{pmatrix} q \\ u \end{pmatrix}, \qquad z_0 = \begin{pmatrix} q_0 \\ u_0 \end{pmatrix}, \qquad f = \begin{pmatrix} Tu \\ M^{-1}h \end{pmatrix}$$

In diesem Fall werden alle Kräfte Null gesetzt:

$$\begin{aligned} W &\equiv 0 \\ \lambda &\equiv 0 \\ \aleph &\equiv \{\} \end{aligned}$$

3.5.2 Numerische Integration: Grundprinzip

Die numerische Integration enthält zwei Hauptschritte, die zu einer Annäherungslösung nach einem bestimmten Gütekriterium führen.

Diskretisierung

Im ersten Schritt wird das Simulationszeitintervall $[0, T]$ in n diskrete Zeitpunkte eingeteilt:

$$\{t_k : \quad k = 0, 1, ..., n\}, \left\{ \begin{array}{c} t_0 = 0 \\ t_0 < t_1 < \cdots < t_n \\ t_n = T \end{array} \right\}$$

Die Integrationsschrittweite des k-ten Integrationsschritts ist dann:

$$\Delta t_k = t_{k+1} - t_k$$

Nährungslösung

Das Prinzip der numerischen Lösung für Differentialgleichungen beruht auf der Annährung der exakten Lösung $z(t_k)$ an jedem Zeitpunkt t_k durch den numerisch errechneten Wert:

$$z_k \approx z(t_k)$$

Die Nährungslösung für den nächsten Zeitpunkt lautet:

$$z_{k+1} = \Phi(z_{k+1}, t_{k+1}, z_k, t_k, z_{k-1}, t_{k-1}, ...)$$

Die Funktion Φ wird anhand des ausgewählten Integrators ausgewertet. Die Exaktheit der Lösung hängt von der Güte der Integrationsverfahren ab, die im nächsten Abschnitt erläutert werden.

3.5.3 Klassifizierung von Gleichungslösern

Es gibt in der Literatur viele Kriterien zur Klassifizierung von Gleichungslösern. Davon sollen hier drei solcher Kriterien zitiert werden, die für die beiden untersuchten Softwarepakete von großer Bedeutung sind:

1. Klassifizierung nach Integrationsart:

   - Explizite Verfahren: Die Funktion Φ wird ausschließlich anhand von bekannten Werte berechnet

     $$z_{k+1} = \Phi(z_k, t_k, z_{k-1}, t_{k-1}, \ldots)$$

- Implizite Verfahren: Diese Gleichungslösungsmethoden beruhen auf der Nutzung des unbekannten Wertes z_{k+1} bei der Berechnung der Funktion Φ

$$z_{k+1} = \Phi(z_{k+1}, t_{k+1}, z_k, t_k, \ldots)$$

2. Klassifizierung nach Stützpunkten:

- Einschrittverfahren (ESV): Die Auswertung der Funktion Φ greift nur auf den zuletzt berechneten Wert z_k zurück:

$$z_{k+1} = \Phi(z_{k+1}, t_{k+1}, z_k, t_k)$$

- Mehrschrittverfahren (MSV): Im Gegensatz zum Einschrittverfahren greift der Gleichungslöser bei der Auswertung von Φ auf Zeitschritte zurück, die mehr als einen Schritt zurückliegen

$$z_{k+1} = \Phi(z_{k+1}, t_{k+1}, z_k, t_k, z_{k-1}, t_{k-1}, \ldots)$$

3. Klassifizierung nach der Art der Schrittweite:

- Feste Integrationsschrittweite: Dieses Kriterium ist einfach zu implementieren. Die Integrationszeitpunkte sind equidistant:

$$\Delta t_1 = \Delta t_2 = \cdots = \Delta t_{k-1} = \Delta t_k$$

- Variable Integrationsschrittweite: Dieses Verfahren berücksichtigt die Dynamik des zu modellierenden bzw. zu simulierenden Systems, daher die hohe Komplexität bei der Implementierung.

3.5.4 Einschrittverfahren

Alle Einschrittverfahren zur Differentialgleichungslösung basieren auf dem Mittelwertsatz, wonach es ein ξ gibt, das die folgenden Bedingungen erfüllt:

$$\begin{aligned} z(t_{k+1}) &= z(t_k) + \Delta t \underbrace{f(z(\xi), \xi)} \\ &= z(t_k) + \Delta t \quad \dot{z}(\xi) \\ \text{und} \qquad t_k \leq &\ \xi \ \leq t_{k+1} \end{aligned}$$

Mit Hilfe des Gleichungslösers wird die unbekannte Variable ξ gesucht, die die beiden Gleichungen erfüllt. Numerische Integrationsmethoden liefern eine Approximation zum Wert von $\dot{z}(\xi)$:

$$F \approx f(z(\xi), \xi)$$

Die Integrationsvorschriften für die Einschrittverfahren unterscheiden sich dann voneinander wie folgt:

- explizites Verfahren: $z_{k+1} = z_k + \Delta t \cdot F(z_k, t_k)$
- implizites Verfahren: $z_{k+1} = z_k + \Delta t \cdot F(z_{k+1}, t_{k+1}, z_k, t_k)$

3.5.5 Explizites und implizites Euler-Verfahren

Im Eulerverfahren berechnet sich z_{k+1} aus dem alten Wert z_k und den ersten Ableitungen der Funktion $z(t)$ jeweils an den Stellen t_k für das explizite und an t_{k+1} für das implizite Eulerverfahren. Die Abbildung 3.6 zeigt die berechneten Werte z_{k+1} mittels beider Eulerverfahren. Die Integrationsvorschriften sind dann:

- explizites Verfahren: $z_{k+1} = z_k + \Delta t \cdot f(z_k, t_k)$
- implizites Verfahren: $z_{k+1} = z_k + \Delta t \cdot f(z_{k+1}, t_{k+1}, z_k, t_k)$

Da viele implizite Verfahren den Wert z_{k+1} voraussetzen, wird eine Approximation dieses Wertes mittels numerischer Verfahren wie die Fixpunktiteration oder die Newton-Raphson-Methode gesucht.
Der Gleichungslöser in ADAMS setzt beispielsweise die Newton-Raphson-Methode ein, die im folgenden Abschnitt beschrieben wird.

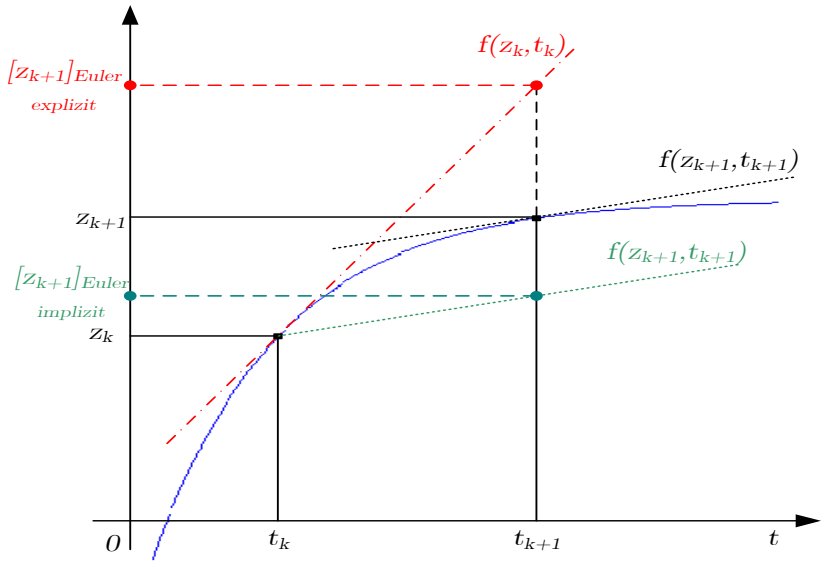


Bild 3.6: Implizites und explizites Eulerverfahren

3.5.6 Newton-Raphson-Methode

Die *Newton-Raphson-Methode* ist ein wichtiger numerischer Algorithmus zur Lösung von Gleichungen der Form $f(z) = 0$. Weiterhin wird er bei Gleichungslösern impliziter Natur verwendet um Schätzwerte für den aktuellen Zeitpunkt zu bestimmen. Im eindimensionalen Raum sucht der Newton-Raphson-Algorithmus die Lösung Z^* der Gleichung:

$$f(z) = 0$$

Die Funktion $f : \mathbb{R} \longrightarrow \mathbb{R}$ soll dabei differenzierbar sein. Aus einer gegebenen Approximationslösung z_0 wird die neue Belegung z_1 nach dieser Vorschrift berechnet:

$$z_1 = z_0 - \frac{f(z_0)}{f'(z_0)}$$

$f'()$ ist hier die Ableitung der Funktion $f()$ an den jeweiligen Stellen.
Die Linearisierung der Funktion f an der Stelle z_0 liefert eine erste Näherung der Funktion gemäß:

$$f(x) \approx f(z_0) + f'(z_0)(z - z_0)$$

bei der eine Lösung z_1 der rechten Hälfte gemäß der folgenden Gleichung berechnet wird:

$$f(z_0) + f'(z_0)(z_1 - z_0) = 0 \quad \Longrightarrow \quad z_1 = z_0 + \frac{f(z_0)}{f'(z_0)}$$

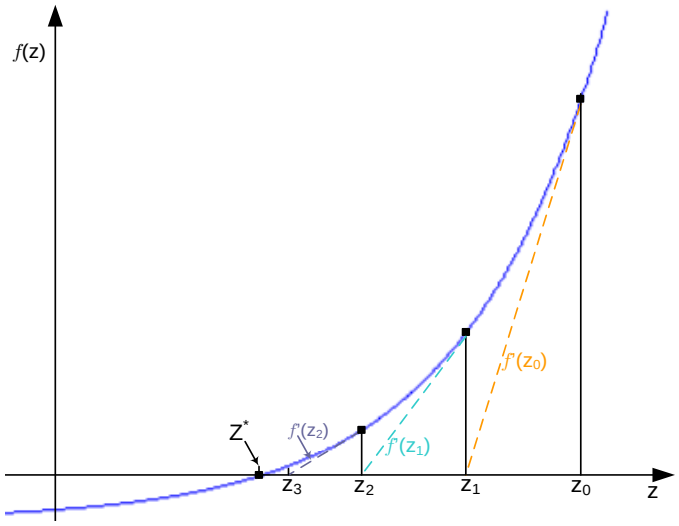


Bild 3.7: Grundlage der Integration: die Newton-Raphson-Methode

Der oben beschriebene Schritt wird so lange iteriert bis sich ein hinreichend geringer Abstand zwischen der gesuchten Lösung Z^* und dem beim $n-$ten Iterationsschritt berechneten Wert z_n eingestellt hat (siehe Abbildung 3.7).
Dieser Ansatz führt zu einer quadratischen Konvergenzrate zum Wert Z^*, mit der Voraussetzung eines „gut“ gewählten Anfangswertes z_0.

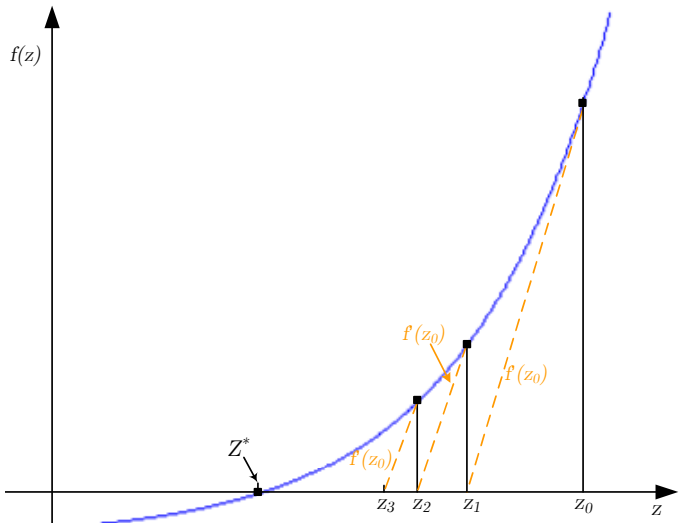


Bild 3.8: Verringerung des Rechenaufwandes: die Newton-Like-Methode

Nachteilig bei dieser Methode ist der große Rechenaufwand innerhalb jedes Iterationsschrittes zur Berechnung von $f(z_k)$ und $f'(z_k)$ und mangelnde Konvergenz bei manchen Funktionen. Um den Rechenaufwand zu reduzieren, wird in einer alternativen Methode die Ableitung

für mehrere Iterationen verwendet und erst nach einer bestimmten Anzahl dieser Iterationen aktualisiert. Die Konvergenz gegen die Endlösung Z^* verläuft in dieser sogenannten *Newton-Like-Methode* linear (siehe Abbildung 3.8).

3.5.7 Verbessertes Euler-Verfahren: Verfahren von Heun

Um zu einer besseren Näherung der Funktion $f(z(\xi), \xi)$ zu gelangen, wird in dem Heun-Verfahrens der Mittelwert der beiden ersten Ableitungen am aktuellen Zeitpunkt t_k und am nächsten zu berechnenden Zeitpunkt t_{k+1} berechnet:

$$\frac{1}{2}(f(z_k) + f(\overline{z}_{k+1}))$$

Der unbekannte Zustand z_{k+1} wird hierbei anhand des expliziten Eulerverfahren an $\overline{z}_{k+1}$ angenähert. Die Berechnung des unbekannten Wertes z_{k+1} verläuft in drei Schritten, die in der Abbildung 3.9 grafisch beschrieben werden. Die mit diesem Verfahren berechnete

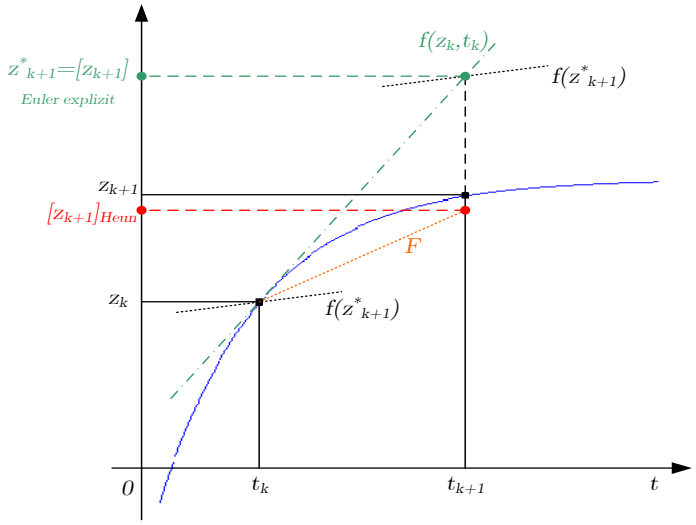


Bild 3.9: Verfahren von Heun

Steigung F am aktuellen Integrationspunkt besteht aus dem Mittelwert der Steigungen an den Punkten t_k und t_{k+1}. Die Werte dieser drei Steigungen werden wie folgt berechnet:

$$\begin{aligned} k_1 &= f(z_k, t_k) \\ k_2 &= f(z_k + \Delta t \cdot k_1, t_k + \Delta t) \\ F &= \frac{1}{2}(k_1, k_2) \end{aligned}$$

3.5.8 Explizites Runge-Kutta-Verfahren

Die *Runge-Kutta-Verfahren* sind als eine Verallgemeinerung des Heun-Verfahren zu sehen. Man kann das Euler-Verfahren als ein *ein*-stufiges Verfahren und Heun als ein *zwei*-stufiges Verfahren ansehen. In seiner klassischen Ausführung wird *Runge-Kutta* als ein *vier*-stufiges Verfahren implementiert.
Die Rechenschritte einer s-stufigen Runge-Kutta-Methode sind dann:

$$\begin{aligned} F &= \sum_{i=1}^{s} b_i k_i \\ k_1 &= f(z_k, t_k) \\ k_i &= f(z_k + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j,\ t_k + c_i \Delta t), \quad i = 2, \ldots, s \end{aligned}$$

Die Koeffizienten eines bestimmten Runge-Kutta-Verfahrens können den Butcher Tabellen entnommen werden (siehe Tabelle 3.2).

Tabelle 3.2: Butcher Tableau: Koeffizienten eines s-stufigen Runge-Kutta-Verfahrens

$$\begin{array}{cccccc} c_1 & & & & & \\ c_2 & a_{21} & & & & \\ \vdots & \vdots & \ddots & & & \\ c_s & a_{s1} & \ldots & a_{s,s-1} & & \\ & b_1 & & & & b_s \end{array}$$

Beispiele für die Koeffizienten eines 1-stufigen und 2-stufigen Runge-Kutta-Verfahrens sind der Tabelle 3.3 zu entnehmen. Eine der bekanntesten Ausführungen ist das *sieben*-stufige Verfahren, bekannt als Dormand und Prince Methode.

Tabelle 3.3: Beispiel für explizites Runge-Kutta-Verfahren: links explizites Euler (1-stufig), rechts Heun (2-stufig)

$$\begin{array}{cc} & \\ 0 & \\ & 1 \end{array} \quad \Bigg\| \quad \begin{array}{ccc} 0 & & \\ 1 & 1 & \\ & 1 & 1 \\ & 2 & 2 \end{array}$$

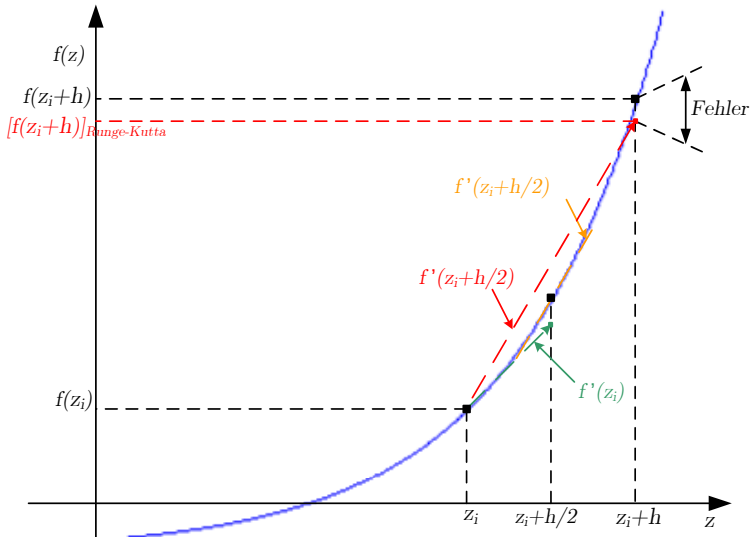


Bild 3.10: Verfahren von Runge-Kutta (2-stufig)

Die Abbildung 3.10 gibt eine grafische Darstellung der *zwei*-stufigen *Runge-Kutta-Methode* wieder. Die Berechnung verläuft hier nach folgendem Plan:

1. Teilung des Zeitintervalls $\Delta t = h$ in zwei gleich große Intervalle $\frac{h}{2}$
2. Berechnung einer Approximation $\overline{f}(z_i + \frac{h}{2})$ anhand des expliziten Euler-Verfahren
3. Anhand des berechneten Werts $\overline{f}(z_i+\frac{h}{2})$ wird an der Stelle $z_i+\frac{h}{2}$ die Ableitung $f'(z_i+\frac{h}{2})$ berechnet
4. Mittels der Ableitung $f'(z_i + \frac{h}{2})$ wird vom Punkt z_i aus der Wert $f(z_i + h)$ geschätzt anhand der Formel: $\overline{f}(z_i + h) = f(z_i) + h \cdot f'(z_i + \frac{h}{2})$

3.5.9 Mehrschrittverfahren

Die grundlegende Idee bei den Mehrschrittverfahren ist die Nutzung von weit in der Zeitachse zurückliegenden Integrationsschritten. Die allgemeine Vorschrift dafür ist:

$$\sum_{j=0}^{m} \alpha_j \, z_{k+1-j} = \Delta t_k \sum_{j=0}^{m} \beta_j \, \dot{z}_{k+1-j}$$

Maßgebend für solche Verfahren ist die Anzahl der Werte m, auf die zurückgegriffen wird, Gewichte α_j der Stützpunkte und Gewichte β_j der Ableitungen an den Stützpunkten. Die

expliziten Verfahren haben den Koeffizienten $\beta_0 = 0$ während bei den *impliziten* Verfahren $\beta_0 \neq 0$ ist.

3.5.10 Adams-Bashforth-Verfahren

Das Adams-Bashforth-Verfahren ist ein explizites Mehrschrittverfahren. Aus der Erkenntnis:

$$z(t_{k+1}) = z_k + \int_{t_k}^{t_{k+1}} \dot{z}(\tau) d\tau$$

bleibt die Bestimmung von $\dot{z}(t)$ offen, die anhand der m-Stützpunkte

$$p_k : (t_k, \dot{z}_k),\ p_{k-1} : (t_{k-1}, \dot{z}_{k-1}),\ \ldots\ , p_{k+1-m} : (t_{k+1-m}, \dot{z}_{k+1-m})$$

berechnet wird.
Als erstes wird das Polynom:

$$P(t) = \sum_{i=1}^{m} L_i\, \dot{z}_{k+1-i}$$

aufgestellt. Es setzt sich aus den Ableitungen an den jeweiligen Stützpunkten und den Lagrange-Polynomen zusammen:

$$L_i(t) = \prod_{j=1, i\neq j}^{m} \frac{t - t_{k+1-j}}{t_{k+1-i} - t_{k+1-j}}$$

Somit können in der oberen Formel die Ableitungen $\dot{z}$ durch die Polynome $P(t)$ ersetzt werden:

$$\dot{z}_{k+1-i} = P(t_{k+1-i}) \qquad i = 1, \ldots, m$$

3.5.11 Backward Differentiation Formula

Das BDF-Verfahren nutzt im Gegensatz zum Adams-Bashforth-Verfahren die Interpolation der $z(t)$-Werte anstelle von $\dot{z}$. Das Interpolationspolynom schreibt sich also wie folgt:

$$P(t) = \sum_{i=0}^{m} L_i\, z_{k+1-i}$$

Für die Lösung des impliziten Systems wird das Newtonverfahren verwendet.

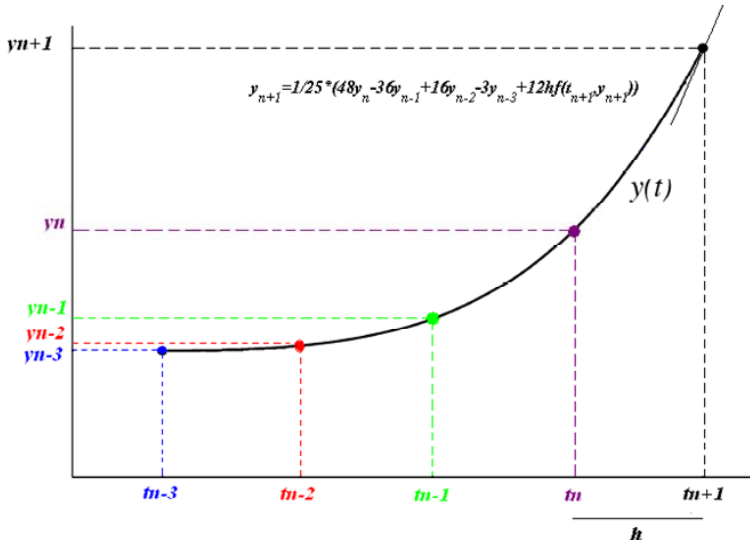


Bild 3.11: Beispiel eines BDF-Verfahrens

3.5.12 Adams-Moulton-Verfahren

Das Adams-Moulton-Verfahren ist ein implizites Mehrschrittverfahren. Es wird genauso berechnet wie das explizite Adams-Bashforth-Verfahren mit der Ausnahme, dass hier das Polynom P mit dem zusätzlichen Stützpunkt $\dot{z}_{k+1}$ berechnet wird. Somit erhält man:

$$P(t) = \sum_{i=0}^{m} L_i \, \dot{z}_{k+1-i}$$

3.5.13 Adams-Prädikator-Korrektor-Verfahren

Dieses Verfahren ist eine Kombination aus den beiden impliziten und expliziten Adams-Verfahren. Es besteht aus zwei Phasen:

1. Prädikatorphase: In dieser Phase wird ein Startwert z_{k+1}^0 mit Hilfe des *expliziten Adams-Verfahren* bestimmt.

2. Korrektorphase: In dieser Phase wird z_{k+1} mit dem *impliziten Adams-Verfahren* und der *Fixpunktiteration* bestimmt.

3.5.14 Schrittweitensteuerung

Beim Simulieren steht man vor der Frage, wie groß die Schrittweite Δt gewählt werden soll, damit die Simulation einerseits schnell genug fortschreitet und anderseits ein Fehler eine vorgeschriebene Toleranz nicht überschreitet. Zu klein gewählte Schritte führen zu einem hohen Rechenaufwand und somit zu großen Simulationszeiten. Dagegen führen zu groß gewählte Schritte zu großen Fehlern und damit zu schlechter Ergebnisgüte.
Um gleichzeitig die vorgeschriebene Fehlertoleranz zu respektieren und die Simulation schnell abzuschließen, wird eine variable Schrittweite verwendet. Die Variation unterliegt der Logik einer Schrittweitensteuerung, bei der grundsätzlich zwei Fälle vorliegen:

- Fall 1: $|Fehler_k| > Toleranz$: in diesem Fall wird der aktuelle Schritt Δt_k verkleinert. Je nach Methode wird ein neuer kleinerer Schritt $\Delta t_{k_{neu}}$ berechnet, mit dem der aktuelle Integrationsschritt dann wiederholt wird.
- Fall 2: $|Fehler_k| < Toleranz$: in diesem Fall wird der aktuelle Schritt Δt_k vergrößert. Je nach Methode wird ein neuer, größerer Schritt $\Delta t_{k_{neu}}$ berechnet, mit dem die aktuelle Integration fortgesetzt wird.

Als Beispiel zur Schrittweitensteuerung dient das *adaptive* Runge-Kutta-Verfahren *RK45*, bei dem der neue zu berechnende Schritt nach der folgenden Formel festgesetzt wird:

$$wenn \quad Fehler > Toleranz \qquad \Longrightarrow h_{new} = h_{aktuell} \left| \frac{Toleranz}{Fehler} \right|^{0.25}$$

$$wenn \quad Fehler < Toleranz \qquad \Longrightarrow h_{new} = h_{aktuell} \left| \frac{Toleranz}{Fehler} \right|^{0.2}$$

4 Analyse der vorhandenen Schnittstelle

4.1 Definition der General State of Equation (GSE)

4.1.1 Allgemeine mathematische Definition

Die *General State of Equation* ist eine Subroutine bzw. eine Funktion, mit deren Hilfe Submodelle und Regelsysteme in einem ADAMS-Modell eingebunden werden können. Die folgenden Gleichungen innerhalb der GSE beschreiben das einzubindende System:

$$\dot{x_c} = f_c(x_c, u, t) \quad x_c(t_0) = x_{c0} \tag{4.1}$$

$$x_{d_{n+1}} = f_d(x_{d_n}, u, t) \quad x_d(t_0) = x_{d0} \tag{4.2}$$

$$y = g(x_c, x_d, u, t) \tag{4.3}$$

Die GSE besteht aus kontinuierlichen Zuständen x_c und diskreten Zuständen x_d. Die kontinuierlichen Zustände x_c sind in der ersten Gleichung als eine ordinäre explizite Gleichung erster Ordnung angegeben. $f_c()$ wird vom Anwender definiert und vom ADAMS-Solver berechnet.
Die zweite Gleichung beschreibt die diskreten Zustände x_d. (5.2) ist eine Differenzengleichung, deshalb wird eine Funktion zur Berechnung der Zeitschritte dazu assoziiert. Innerhalb einer Periodendauer bleiben die Werte $x_d()$ unverändert. x_{d_n} ist dabei die abgekürzte Schreibweise von $x_d(t_n)$.
$g()$ definiert in der dritten Gleichung den Ausgang y und wird kontinuierlich über die gesamte Simulationszeit hindurch abgetastet.

4.1.2 Typen der GSE

Wenn die erste Gleichung nicht vorhanden ist, dann wird die GSE als rein *diskret* klassifiziert. Ist dagegen die zweite Gleichung nicht vorhanden, wird die GSE als rein *kontinuierlich*

klassifiziert. Die GSE wird als *abgetastetes System* eingeordnet, wenn beide Gleichungen vorhanden sind. Sie enthält **keine** *internen Zustände*, falls beide Gleichungen nicht vorhanden sind.
Um ein besseres Verständnis einer GSE zu erlangen, genügt eine klare Definition der beiden ersten Typen und deren Unterschiede.

Kontinuierliche GSE

Nehmen wir die Gleichung (5.1) wieder unter die Lupe. Für ein System n-ter Ordnung beinhaltet der kontinuierliche Zustand x_c n Elemente. In der Matrixschreibweise hat sie also die Dimension $[n \times 1]$. u definiert den Eingangsvektor. Bei m Eingängen hat u die Größe $[m \times 1]$. Das Gleiche gilt für den Ausgangsvektor y, der als eine Spaltenmatrix mit der Dimension $[p \times 1]$ geschrieben wird.

Ein nicht lineares System 2-ter Ordnung sei nun beschrieben durch die folgende Differentialgleichung:

$$\ddot{y} + 2D\omega_n y\dot{y} + \omega_n^2 y \quad = \quad Ku \tag{4.4}$$

Nach dem Umschreiben in die Zustandsgleichungen bekommt man:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} \quad = \quad \begin{pmatrix} x_2 \\ -2D\omega_n x_1 x_2 - \omega_n^2 x_1 + Ku \end{pmatrix} \tag{4.5}$$

$$y \quad = \quad x_1 \tag{4.6}$$

Die Gleichungen (5.1) oder (5.5) werden mit dem ADAMS-Gleichungslöser gelöst. Die vorhandene GSE setzt nach [67] ein stetige, funktionale Beziehung zwischen den beiden Hälften der Gleichungen voraus. Wird der Ausgang der GSE wieder von der Modellstrecke in ADAMS verwendet, so müssen die Gleichungen (5.3) oder (5.6) kontinuierlich sein. Eine hohe Differenzierbarkeit hilft den Solver, die Gleichungen mit höheren Effizienz zu lösen.

Diskrete GSE

Die diskrete GSE besteht aus Differenzengleichungen. Nach (5.2) stellt x_d für ein diskretes System n-ter Ordnung einen Zustandsvektor der Größe $[n \times 1]$ dar. u und y haben hier die

gleiche Bedeutung wie bei der diskreten GSE.
Der Hauptunterschied zwischen beiden GSEs liegt darin, dass das digitale System in Abtastpunkten und nicht kontinuierlich über die Systemzustände agiert. Die Dynamik des abgebildeten Systems wird dann anhand rekursiver, algebraischer Gleichungen (Differenzengleichungen) gemäß (5.2) dargestellt.

Je nach System erfolgt die Abtastung entweder in fester Schrittweite T oder in variabler Schrittweite, die sowohl von der Zeit als auch vom Systemzustand abhängt. Zwischen den Abtastzeitpunkten werden die Werte der Zustände mit dem Halteglied konstant gehalten. Dieses Glied entspricht einen mathematischen Modell eines D/A-Wandlers [70].

$$G_H(s) = \frac{1 - e^{-Ts}}{s}$$

Wie jedes digitale System muss die Abtastung der Shanon-Theorie unterliegen: die Abtastfrequenz ω_s muss mindestens doppelt so groß sein wie die höchste Frequenz im System ω_{max}. Um sanfte Kurvenverläufe der Ergebnisse zu bekommen, muss jedoch die folgende experimentelle Ungleichung beachtet werden:

$$20 < \frac{\omega_s}{\omega_{max}} < 40$$

4.2 Aufruf und Struktur der GSE in ADAMS

4.2.1 Aufruf

Die GSE wird vom Solver als eine Subroutine mit bestimmten Eigenschaften aufgerufen. Subroutinen in ADAMS ermöglichen die allgemeine und effiziente Gestaltung eines Modells, da sie frei programmierbar sind. Somit könnte man die Funktionalitäten des Solvers erweitern. Dies ist besonders wichtig bei GSEs, die viele „logische“ Elemente enthalten.

Um eine GSE im Modell zu erzeugen, werden im Allgemeinen folgende Schritte durchgeführt:

1. Erstellen einer Bibliothek.
2. Festlegung der zu importierenden Bibliothek für das *Update* der GSE während der Simulation. Die GSE sollte genau definierte Ein- und Ausgänge haben, sowie konfigurierbare Parameter.

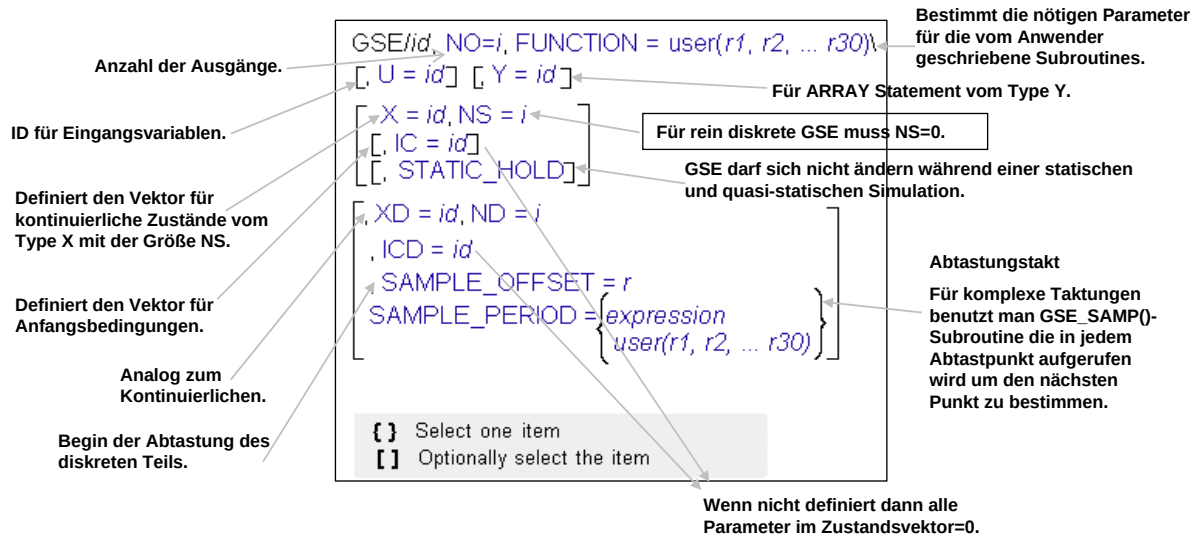


Bild 4.1: Aufruf einer GSE innerhalb eines ADAMS-Modells

3. Fehlercheck um die Übereinstimmung zwischen dem System und der zu importierenden GSE zu überprüfen.

4. Generierung eines Eintrags in der Modelldatenbank in Form eines *property file* (siehe dazu die Abbildung 2.7) mit den relevanten Daten aus dem importierten Simulink-Modell.

Während der Simulation rechnet die geladene Bibliothek die Ableitungen der Systemzustände aus und liefert ihrer Ausgänge an den ADAMS-Gleichungslöser, der diese Ableitungen dann integriert.

4.2.2 Struktur einer GSE

Abbildung 4.2 zeigt den modularen Aufbau einer GSE. Die GSE enthält *vier* Vektoren:

IC-Vektor (Initial Condition) zum Speichern der Anfangswerte.

Zustandsvektor zum Speichern der systeminternen Zustandsgrößen.

Eingangsvektor zum Zwischenspeichern der Eingangswerte der GSE.

Ausgangsvektor Nach der Berechnung der Zustandswerte werden hier die Systemzustände ausgegeben.

Mit der Funktion *GSE_Update* werden die Zustandsgleichungen anhand der drei Parametersätze Anfangswertvektor, Eingangsvektor und die Ausgaben der importierten DLL[1] für die aktuelle Abtastzeit aufgestellt. Die berechneten Zustandsgrößen werden im Zustandsvektor gespeichert. Aus dem Eingangsvektor und der Integration des Zustandsvektors errechnet die Funktion *GSE_Output* für die aktuelle Simulationszeit die Ausgangsvariablen.

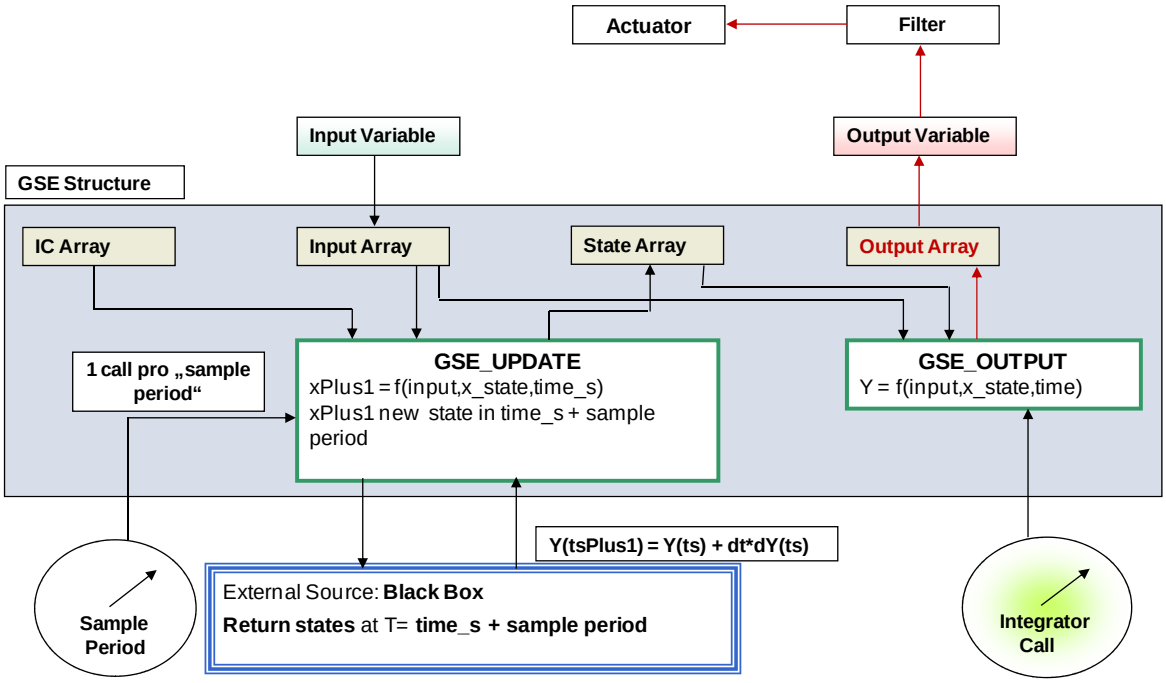


Bild 4.2: Struktur einer GSE innerhalb eines ADAMS-Modells *(Quelle MSC.Software)*

4.3 Funktionen in der Schnittstelle

Um die Übereinstimmung mit dem von der GSE unterstützen API[2]-Standard zu gewährleisten, soll der Quellcode des zu importierenden Systems die in der Tabelle 4.1 aufgeführten Funktionen enthalten.

[1] dynamic link library: Im Bild 4.2 als Black Box bezeichnet.
[2] Application Programming Interface

Tabelle 4.1: Schnittstellenfunktionen

Reihenfolge	Funktion	Beschreibung
1	char* RTT_GetModelName()	Name des zu importierenden regelungstechnischen Systems. Er muss mit dem Namen der dll übereinstimmen. Es muss statisch oder global deklariert werden, so dass der GSE-Treiber die dll einwandfrei laden kann.
2	double RTT_GetStepSize()	Bestimmt die maximale Schrittweite für eine gute Auflösung der abgetasteten Ausgänge.
3	int RTT_GetNumU()	Anzahl der Eingänge
4	int RTT_GetNumY()	Anzahl der Ausgänge
5	int RTT_GetNumX()	Anzahl der kontinuierlichen Zustände
6	int RTT_GetNumDiscX()	Anzahl der diskreten Zustände. Für *sampled* Systeme soll der Wert 1 zurückgegeben werden.
7	int RTT_GetNumParameters()	Anzahl der Parameter der GSE
8	real_T* RTT_GetParameter()	Liste der realen Parameter für die GSE
9	char* RTT_GetParameterName(int i)	Gibt die Namen der Parameter in Form von Zeichenketten zurück
10	void RTT_SetParameter(int numpara, double* para)	Ermöglicht die Änderung der Parameter
11	double *RTT_GetDX(double time, double *U, double *X, double *value)	Gibt die Ableitungen eines Zustandsvektors bei gegebenem Eingang U, Zustand X und Zeit zurück.
12	double *RTT_GetY(double time, double *U, double *X, double *value)	Rückgabe der Ausgangswerte.
13	double *RTT_GetDiscY(double time, double *U, double *X, double *value)	Ausgang an einem Abtastpunkt.
14	double *RTT_GetDiscX(double *DiscX)	Erfassung der diskreten Zustände.
15	double RTT_tNext(double time)	Erfassung der Größe des aktuellen Simulationsschrittes.

4.4 Anforderungen an die GSE

Um die Potentiale der GSE-Schnittstelle maximal auszunutzen, muss die Simulation innerhalb ADAMS mit einem Integrator durchgeführt werden, der eine variable Schrittweite anbietet wie z.B. Gear, Dassl, Wielenga oder RungaKutta45. Die variable Schrittweite erlaubt auf der einen Seite ein „schnelles“ Rechnen bei kontinuierlichem und niederfrequentem Systemverhalten; und auf der anderen Seite ein „präzises“ Rechnen bei den dynamischen Vorgängen.
Bei der Kopplung von Modellen aus Matlab/Simulink an ADAMS durch die GSE liegen dem Gleichungslöser die in der DLL beschriebenen Gleichungen nicht in offener Form vor. Sie werden als *Black-Box* behandelt und daher kann das angewandte Solververfahren nicht prädikativ die Dynaxität[3] des Systems behandeln. Außerdem kann der ADAMS-Gleichungslöser nicht mehr als einen Schritt in die Vergangenheit rechnen. Wenn ein Toleranzbereich für die Ergebnisse beim letzten Schritt überschritten ist, wird der aktuelle Datensatz verworfen und erneut berechnet.

Man braucht also einen Mechanismus, der die fehlende Information über die Dynaxität des *Black-Box*-Systems in der GSE kompensiert. Dabei wird die Fähigkeit von ADAMS-Solver genutzt, einen aktuellen Rechenschritt zu verwerfen und neu zu berechnen. Dies verbessert die Adaptation des verwendeten Gleichungslösers, so dass das Ziel vom *schnellen* und *präzisen* Rechnen besser erreicht werden kann.

Eventproblem in der vorhandenen Schnittstelle

Der gewünschte Mechanismus wird dann besonders wichtig, wenn die GSE-interne Variablen Unstetigkeiten und Diskontinuitäten (nach der Definition im *Unterkapitel 2.7.2*) in ihren Verläufen aufweisen. Es können dann große Diskrepanzen in den Ergebnissen entstehen.

[3] Dynamik-Komplexität

5 Konzept: „Zero-Crossing-Handshake“ (ZC-HaSh)

5.1 Erläuterung

In diesem Kapitel wird ein Lösungskonzept vorgeschlagen, mit dessen Hilfe das im Unterkapitel 4.4 beschriebene Ziel realisiert werden kann. Hierbei wird die Schnittstelle GSE in ihrer Funktionalität so erweitert, dass Unstetigkeiten und Diskontinuitäten im importierten Black-Box-Modell besser vom angewandten ADAMS-Gleichungslöser erfasst werden können. Erstrebenswert ist eine integrierte Simulation auf Modellebene bei der ein adaptiver Gleichungslöser „schnell“ die gesamte Simulation durchführt, ohne Nachteile bei der Ergebniseffizienz aufzuweisen.

Die Idee bei diesem Konzept liegt darin, die Unstetigkeiten und Diskontinuitäten aus dem zu exportierenden MATLAB/Simulink Modell über zusätzlich erzeugte Variablen innerhalb des Ausgangsvektors der GSE an den ADAMS-Gleichungslöser weiterzuleiten. Der Gleichungslöser verwendet diese Information bei der Berechnung des adaptiven Fortschrittes der Simulation. Aus diesem Grund habe ich der Methode den Namen „**Z**ero **C**rossing **Ha**nd**sh**ake“ gegeben. Im weiteren Verlauf der Arbeit wird jedoch öfter die Abkürzung *ZC-HaSh* verwendet.

5.2 Behandlung von Diskontinuitäten in Matlab/Simulink

Bei der Simulation dynamischer Systeme innerhalb Simulink überprüft ein adaptiver Gleichungslöser die Ableitungen der Zustandsvariablen des Systems nach Diskontinuitäten in jedem Zeitschritt. Die dabei verwendete Technik heißt *Zero-Crossing Detection* (ZCD).
Wenn eine Diskontinuität detektiert ist, versucht der Gleichungslöser mit einer hohen Genauigkeit den Zeitpunkt zu bestimmen, an dem diese Unstetigkeit sich ereignet hat. Um

dieses Ziel zu erreichen, sind mehrere zusätzliche Simulationsschritte vor und nach diesem Zeitpunkt nötig.

Im Allgemeinen stimmen die Diskontinuitäten zeitlich mit einem wichtigen Ereignis im Bewegungsverlauf eines dynamischen Systems überein. In dem bekannten Beispiel eines fallenden Balles stellt der Zeitpunkt des Auftreffens auf den Boden eine solche Diskontinuität dar. Ein bedeutsamer Wandel im dynamischen System findet an diesem Punkt statt, nämlich ein Richtungswechsel in der Bewegung des Balles. Sollte dieser Zeitpunkt nicht mit einer hohen Genauigkeit berechnet sein, so erhält man falsche Simulationsergebnisse bezüglich dieses Systems.
Um falsche Rückschlüsse zu vermeiden, sollte ein Integrationsschritt möglichst im selben Zeitpunkt des Ereignisses stattfinden. Ein Simulator, der ausschließlich den Gleichungslöser zur Bestimmung der Simulationsschritte heranzieht, kann diese Anforderung nicht erfüllen (siehe [24]). Ein Gleichungslöser mit einer festen Schrittweite könnte diese Anforderung erfüllen, indem die Schrittweite sehr klein gewählt wird. Jedoch muss man mit erheblichen Nachteilen bezüglich der Ausführungszeit der Simulation rechnen. Ein Gleichungslöser mit einer variablen Schrittweite kann dagegen das Problem besser lösen, indem er die Schrittweite bei einer langsamen Veränderung der Variablen vergrößert und sie bei schnellen Veränderungen der Variablen verkleinert. Obwohl der adaptive Gleichungslöser diese Anforderung mit dem eigenen Verfahren erfüllen kann, leidet die Ausführungszeit in der Nähe der Diskontinuität. Jede Suche nach solchen Punkten erfordert je nach Simulation eine mehrmalige Verringerung der Schrittweite und eventuell das Wiederholen einiger Schritte, die das Konvergenzkriterium des verwendeten Integrationsverfahrens nicht erfüllen. Abhilfe schafft hier die im folgenden beschriebene Technik: ZCD(*Zero-Crossing-Detection*).

5.2.1 Funktionsweise der ZCD

Innerhalb eines Simulink-Blocks können Signale zwecks Überprüfung auf Diskontinuitäten registriert werden. Diese Signale werden *Zero-Crossing*-Signale genannt und sind eine Funktion der Zustandsvariablen, innerhalb derer die Diskontinuitäten entdeckt werden sollen. Die besonderen Eigenschaften dieser Signale liegt darin, dass sie einen Nulldurchgang produzieren und zwar genau dann, wenn das *Event* sich ereignet. Am Ende jedes Integrationsschritts fragt der Gleichungslöser in Simulink alle Blöcke mit registrierten ZC-Variablen nach einer Aktualisierung dieser Variablen. Im Anschluss überprüft Simulink, ob eine dieser Variablen ihr Vorzeichen seit dem letzten Schritt verändert hat. Solche Vorzeichenwechsel deuten auf eine Diskontinuität hin, die innerhalb des letzten Schritts geschehen ist.
Beim Entdecken eines Nulldurchgangs innerhalb eines Signals interpoliert Simulink zwischen

dem aktuellen und dem alten Wert dieser Variablen, um den Zeitpunkt des Nulldurchgangs abzuschätzen. In diesem Fall inkrementiert Simulink die Schritte abwechselnd bis zum Nulldrchgang und drüber. Um undefinierte Zustände zu vermeiden, wird der Zeitpunkt des Nulldurchgangs nicht als Simulationspunkt verwendet.

5.2.2 Implementierung

Ein Beispiel für einen Simulink-Block mit einer ZCD ist die Sättigungsfunktion. Die Zustandsereignisse innerhalb des Blocks sind:

- Eingangssignal erreicht den oberen Sättigungswert.
- Eingangssignal erreicht den unteren Sättigungswert.
- Eingangssignal unterschreitet den oberen Sättigungswert.
- Eingangssignal überschreitet den unteren Sättigungswert.

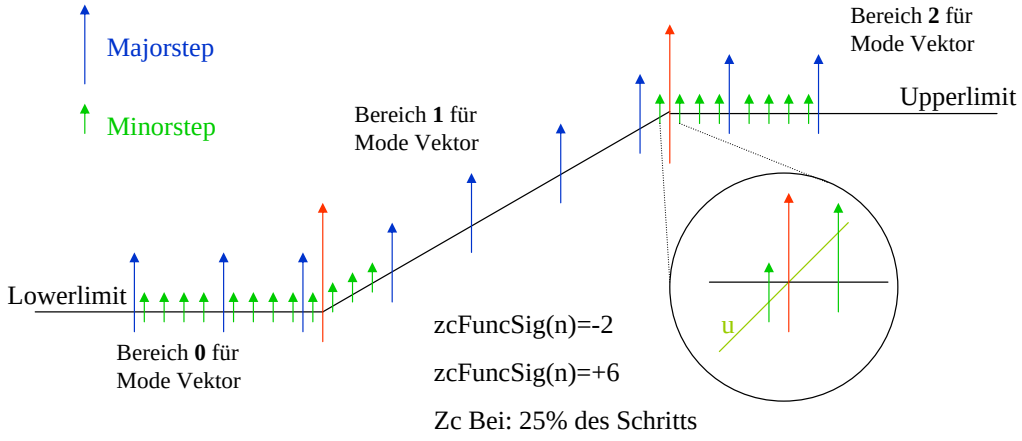


Bild 5.1: *Eventhandling*-Mechanismus in Simulink

Das Signal zur Erkennung des Nulldurchgangs im oberen Sättigungsbereich wird als:

$$zcSignal = Upperlimit - u$$

definiert. Dabei ist u das Eingangssignal zum Block; $Upperlimit$ ist der obere Sättigungswert. Gleiches gilt für das Signal im unteren Sättigungsbereich mit:

$$zcSignal = u - Unterlimit$$

Im allgemeinen werden zero-crossing-Signale mit einem Attribut versehen, mit dem die Richtung des Signals festgelegt werden kann: steigend, fallend oder beides. Somit ist es möglich Nulldurchgänge in beiden Richtungen mit demselben Signal zu entdecken.

5.2.3 Simulink-Blöcke mit ZCD

In Simulink gibt es Blöcke, die ein oder mehrere interne ZC-Signale besitzen. Sie sind in der Tabelle 5.1 aufgeführt.

Für andere Systeme besteht die Möglichkeit, mit Hilfe eines *Hit-crossing*-Blocks die Diskontinuitäten zu erkennen.

5.3 Behandlung von Diskontinuitäten in ADAMS

5.3.1 Definition von Sensoren

Die Hauptaufgabe der Sensor-Funktion in ADAMS ist die Triggerung von Aktionen, wenn ein Ereignis eintritt.

5.3.2 Sensoraktionen

Es gibt vier Arten von getriggerten Aktionen:

- **Beenden der Simulation:**
  Hiermit kann die Simulation abrupt beendet werden und zwar genau dann, wenn ein vordefiniertes Event geschieht, zum Beispiel wenn eine Variable einen bestimmten Schwellenwert erreicht hat.
- **Änderung von Simulationsparametern:**
  Wenn ein Ereignis geschieht, dann ändert der Sensor die Simulationsschrittweite. So kann die Zeit vor der Kollision zweier Objekte gut aufgelöst werden.
- **Änderung der Eingänge während einer Simulation:**
  Die Eingaben einer Simulation, wie zum Beispiel Fahrmanöver, können hiermit bei laufender Simulation geändert werden.

Tabelle 5.1: Simulink-Blöcke mit ZCD Quelle: Simulink Help; Übersetzung aus dem engl.

Blöcke	Anzahl der zc-Signale	Beschreibung der ZC-Funktionalität. ZC-Signale werden verwendet um:
Abs	1	Nulldurchgänge des Eingangssignals in beiden Richtungen zu entdecken.
Backlash	2	zu entdecken, wenn der untere bzw. der obere Grenzwert unterschritten bzw. überschritten ist.
Dead Zone	2	zu entdecken, wenn die *Dead-Zone* betreten und verlassen wurde.
Hit Crossing	1	zu erkennen, wenn das Eingangssignal die vorgegebene Schwelle überquert.
Integrator	1 oder 3	das Reset-Signal zu erkennen, wenn das Resetport vorhanden ist. Sollte der Integrator begrenzt sein, dann gibt es drei ZC-Signale: Erreichen des oberen und unteren Sättigungswertes und Verlassen der Sättigungszone.
MinMax	1	für jedes Ausgangssignal zu prüfen, welches Eingangssignal das aktuelle Minimum oder Maximum ist.
Relay	1	den Umschaltpunkt zu erkennen.
Relational Operator	1	Ausgangsänderungen zu erkennen.
Saturation	2	das Erreichen und Verlassen des oberen bzw. unteren Sättigungswert zu erkennen.
Sign	1	Nulldurchgänge zu erkennen.
Step	1	Zeitschritte zu erkennen.
Subsystem	1 und 1 optional	das Triggersignal für den Freigabeport und für den Triggerungsport zu erkennen.
Switch	1	zu entdecken, wenn Änderungen geschehen.

- **Änderung einer Modelltopologie:**
  Hiermit können z.B. Verbindungen gelöst werden, wenn ein bestimmter Kraftschwellenwert erreicht ist.

Im Allgemeinen kann man die Aktionen in ADAMS in zwei Hauptgruppen unterteilen: Standardaktionen und spezielle Aktionen.

Standardaktionen

Beim Eintreffen eines *Events* kann ADAMS folgende Aktionen auslösen:

- Erzeugung eines neuen Ausgabepunktes. So kann das Event besser aufgezeichnet werden.
- Veränderung der Ausgabeintervalle: die Zeit zwischen den nachfolgenden Ausgabepunkten wird dann definiert.
- Beenden des aktuellen Simulationsschrittes. Im Anschluss kann die Simulation beendet werden oder mit einem *Simulationsskript* fortgesetzt werden.

Spezielle Aktionen

Sie sind für „tiefere“ Eingriffe geeignet und daher gute Werkzeuge für das Debugging der Simulationen.

- **Einstellung der Integrationsschrittweite:**
  Diese Änderung gilt nur für den nächsten Integrationsschritt, es sei denn, der Integrator wird neu mitgestartet.
- **Integrator-Neustart:**
  Der Integrator wird neu gestartet und die Integrationsordnung[1] wird auf *eins* reduziert. Wenn dies mit einer Einstellung der Integrationsschrittweite kombiniert wird, dann wird diese Schrittweite für den Rest der Simulation übernommen. Umso wichtiger wird diese Aktion je mehr der verwendeter Gleichungslöser auf „ältere“ Werte zurückgreift, da der Gleichungslöser ab diesem Punkt alle Werte aus der „Vergangenheit“ ignoriert.
- **Refaktorisierung der Jakobimatrix:**
  ADAMS/Solver generiert erneut die Jakobimatrix für die Elementenbeschreibung. Dies verhilft dem Gleichungslöser, effizientere Ergebnisse zu erzielen und die Simulation kann dadurch robuster verlaufen.
- **Speicherung des Zustandsvariablenvektors:**
  Diese Aktion dient dem Debugging. Der komplette Zustandsvektor wird in ein File innerhalb des Simulationsordners geschrieben.

[1] Die Integrationsordnung ist eine Bezeichnung der Ordnung der Polynome, die in der Lösung vorkommen

5.3.3 Triggerung eines Sensors

Der Sensor wird ausgelöst, wenn ein spezifizierter Schwellenwert erreicht ist. Eine Vergleichsvorschrift zwischen dem aktuellen Wert und dem Zielwert, sowie eine zulässige Toleranz des Zielwertes werden ebenfalls wie im Folgenden vorgegeben.

Toleranz

Da eine Funktion selten einem Zielwert genau entspricht, wird eine zulässige Fehlertoleranz spezifiziert. An jedem Integrationsschritt vergleicht ADAMS/Solver den aktuellen Wert der Funktion mit dem Zielwert. Die vorgegebene Fehlertoleranz wird dabei berücksichtigt.

Vergleichsvorschrift

Mit dieser Vorschrift wird der Typ des Vergleiches festgelegt, den der ADAMS/Solver verwendet um die Sensorfunktion einzuleiten. Tabelle 5.3 gibt eine Übersicht über die Vergleichsmöglichkeiten innerhalb der Sensorfunktion und wie die Fehlertoleranz hierbei berücksichtigt wird.

Tabelle 5.3: Vergleichsarten innerhalb einer Sensorfunktion

Vergleich	Leitet die Aktion, ein wenn die Funktion den Wert x annimmt
Gleich	$(Zielwert - Toleranz) \leq x \leq (Zielwert + Toleranz)$
Größer gleich	$x \geq (Zielwert - Toleranz)$
kleiner gleich	$x \leq (Zielwert + Toleranz)$

5.3.4 Erfassung der ZC-Events in ADAMS

Um Diskontinuitäten in ADAMS zu entdecken, werden zwei Sensoren für jedes Event generiert, mit deren Hilfe die Integrationsschrittweite nur in der Nähe der Diskontinuität reduziert wird.
$f(t)$ sei eine Funktion der Simulationszeit, die auf mögliche Diskontinuitäten während der gesamten Simulationszeit geprüft werden soll. Die zwei verwendeten Sensoren werden dann so definiert:

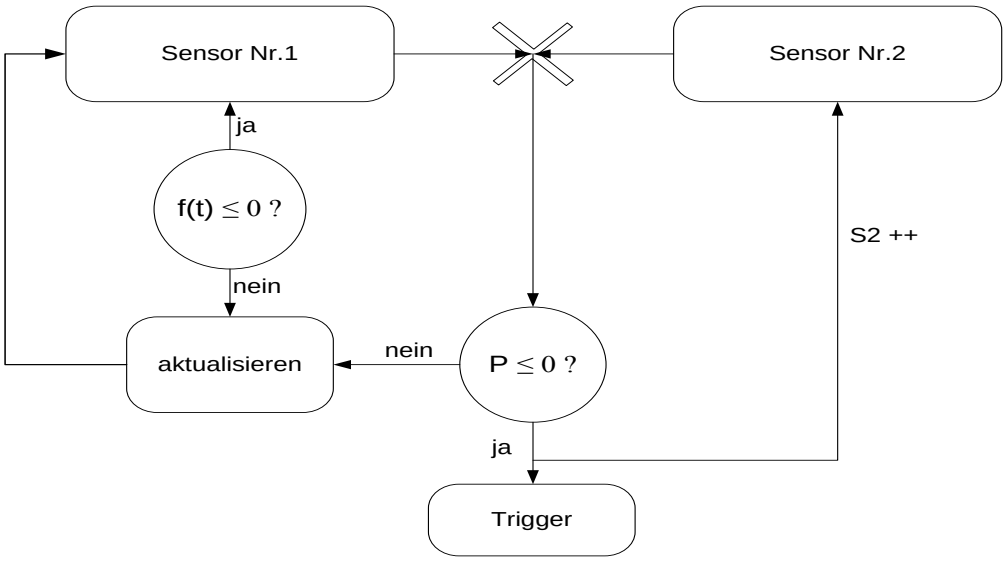


Bild 5.2: Struktur eines ZCD-Mechanismus in ADAMS

- Sensor Nr.1: speichert den Wert $f(k-1)$. Die Ausgabe dieser Sensorfunktion ist eine bool´sche Variable $S_1(t)$, die den Wert 1 erst dann annimmt wenn $f(t) \leq 0$; sonst ist $S_1(t) = 0$.

- Sensor Nr.2: speichert das Produkt P aus dem Sensor Nr.1 Wert und dem aktuellen Wert der bewerteten Variable $f(k)$:

$$P = S_1 \cdot f(k)$$

Die Abbildung 5.2 zeigt die Logik, wie die beiden Sensoren zur Reduzierung der Schrittweite verwendet werden.

In Abbildung 5.3 kann man erkennen, dass das Produkt P erst dann negativ wird, wenn ein Nulldurchgang innerhalb des aktuellen Integrationsintervalls $T = [k-1, k]$ vorliegt und $f(k-1)$ und $f(k)$ unterschiedliche Vorzeichen in diesem Intervall aufweisen. Für das Debugging wird der Wert der Sensorfunktion S_2 jedes Mal um 1 inkrementiert, wenn eine Nulldurchgang erfasst wird. Die Toleranz wird in der Größenordnung $10^{-8}s$ gewählt. Die dabei ausgelösten Aktionen sind ein zusätzlicher Integrations- und Ausgabepunkt.

Tabelle 5.4 zeigt, wie die beiden Sensoren für jedes Signal konfiguriert werden sollen.

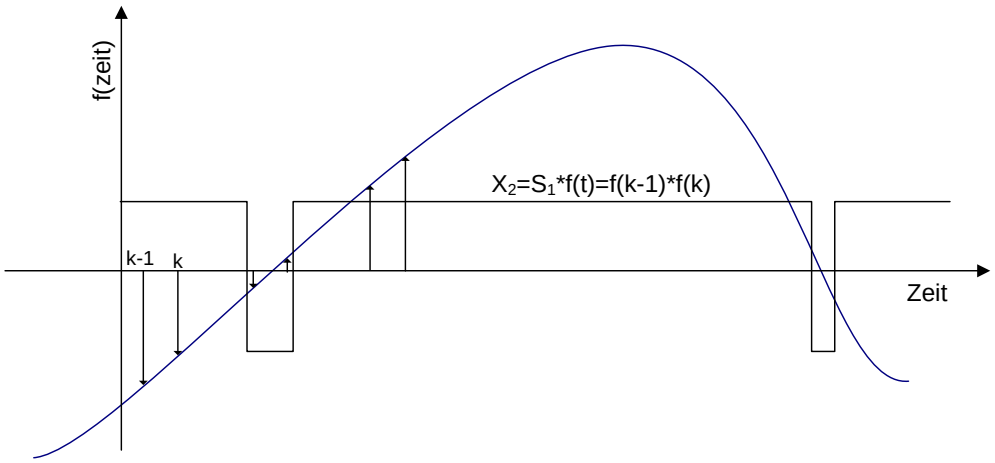


Bild 5.3: *Eventhandling*-Mechanismus in ADAMS

Tabelle 5.4: Konfiguration der Sensorfunktionen

Name	Sensor Nr.1	Sensor Nr.2
Funktion	$f(k-1)$	$P = f(k) \cdot S_1$
Bedingung	$f(k-1) \geq 0$?	$P \leq 0$?
Ergebnis	wenn $f(k-1) \geq 0$ dann $S_1 = 1$ sonst $S_1 = 0$	wenn $P \leq 0$ dann $S_2 + +$
Standard Aktionen	keine	Zusätzlicher Ausgabepunkt
Spezielle Aktionen	keine	Zusätzlicher Integrationspunkt
Toleranz	$10^{-8}s$	$10^{-8}s$

5.4 Umsetzung von ZC-HaSh

Um die ZC-HaSh zu implementieren, werden die normalerweise internen ZC-Signale in den entsprechenden Simulink-Blöcken nach „außen“ als Ausgänge geschrieben. Anschließend wird für jedes Signal eine ZC-Überprüfung mittels zwei Sensoren in ADAMS vorbereitet. Um die Ergebnisse zu veranschaulichen, wird ein selbst geschriebenes Programm verwendet, das die Integrationspunkte für die jeweilige Variable während der Simulation in einem MAT^2-File speichert.
Im Folgenden werden diese Schritte genauer beschrieben.

[2]Ein Filetyp, der eine Matrix-ähnliche Struktur hat.

5.4.1 TAB-Programm

Es handelt sich um ein $C++$-Programm, das die Ergebnisdaten aus der Simulation herausliest und diese in ein File schreibt. Das gespeicherte File wird dann in MATLAB/Simulink in Form von Kurven dargestellt. Das Besondere an diesen Kurven ist die Darstellung der Kurven $\overline{f}$ mittels:

$$Variablenwert = \overline{f}(Integrationspunkt)$$

Dies ermöglicht eine klare Aussage über das Integrationsverhalten des Gleichungslösers, denn die Kurven, die ADAMS mit dem Veranschaulichungstool *Postprocessing* darstellt, beruhen auf festen Ausgabepunkten mit festen Abständen, die aus der Interpolation der Integrationspunkte ausgelesen werden.

5.4.2 Änderungen in MATLAB/Simulink

Um zusätzliche Signale vom Simulink-Modell in die GSE zu schreiben muss man eine Änderung am generierten Code vornehmen.

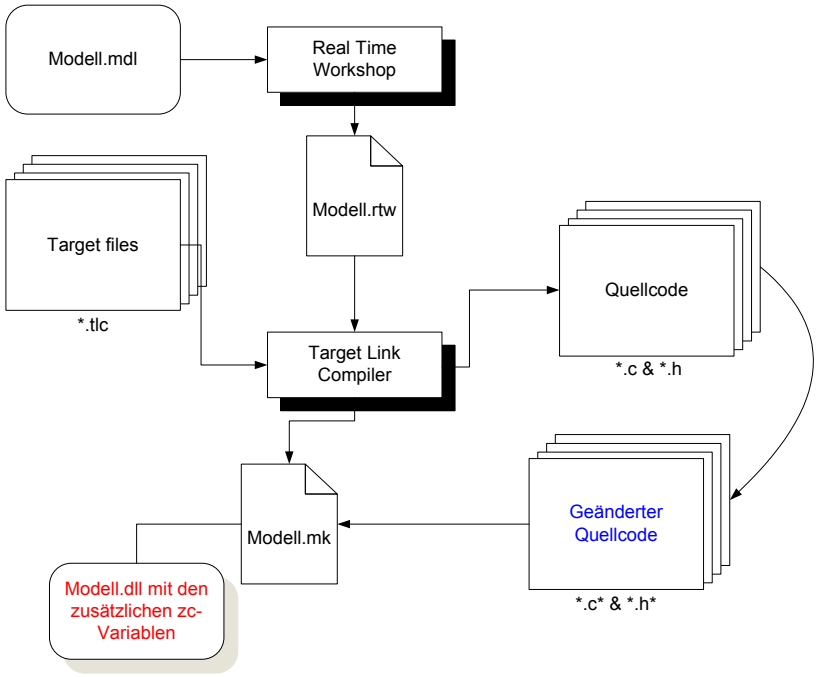


Bild 5.4: Erste Testphase des ZC-HaSh Konzeptes: Geänderter Quellcode

1. Testphase:

In der ersten Testphase des Konzeptes wird statt einer DLL der gesamte Code des Modells mittels RTW (Real Time Workshop) erzeugt (siehe Abbildung 5.4). Zusätzlich wird ein Makefile generiert, das die Kompilierungsvorschriften für die DLL enthält. Der Code wird dann so verändert, dass die zusätzlichen Variablen als Modellausgänge auftauchen. Im Anschluss werden die geänderten *Code-Files* mit Hilfe des Makefiles zu einer DLL kompiliert und schließlich ins ADAMS-Modell eingebunden.

2. Testphase:

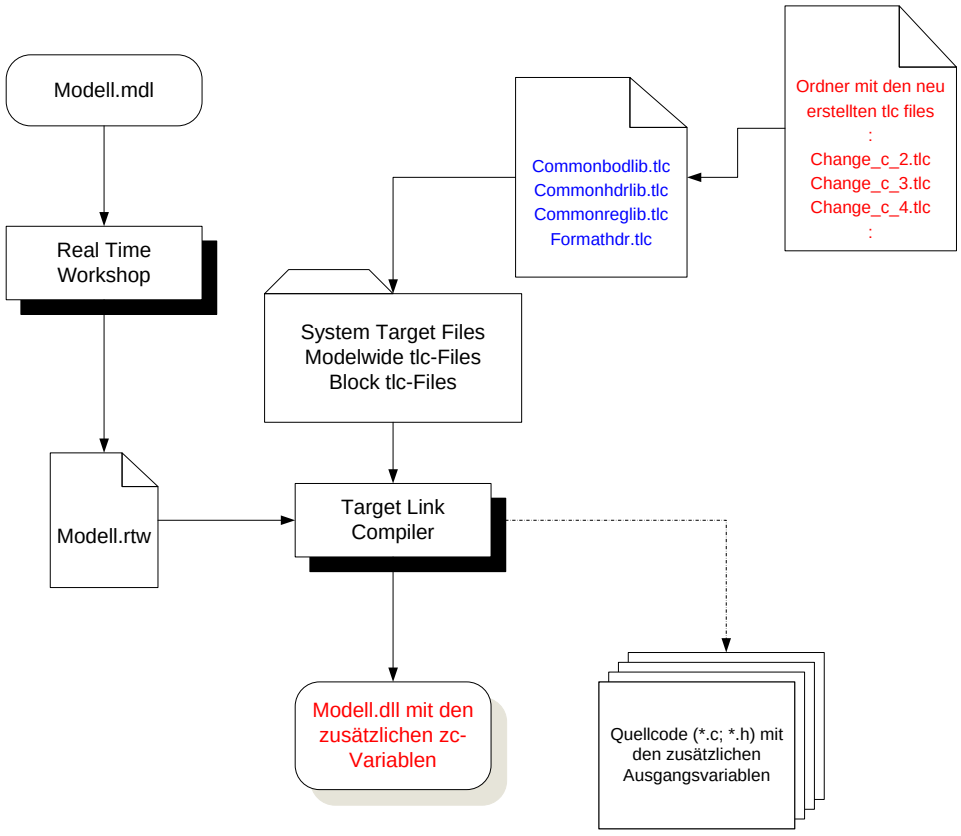


Bild 5.5: Zweite Testphase des ZC-HaSh Konzeptes: automatische Generierung der zusätzlichen Ausgänge

In der zweiten Testphase des Konzeptes wird der Vorgang automatisiert. Dafür ist eine Änderung der TLC (Target Language Compiler)-Files nötig. Dazu wird eine Reihe von Files erstellt, die an bestimmten Stellen in die entsprechenden TLC-Files eingeführt werden. Ab-

bildung 5.5 zeigt die Struktur dieser Änderung für zwei Beispielblöcke: den *Saturation*- und den *Dead-Zone*-Block.

5.5 Simulationsablauf

Um die ZC-HaSh-Methode zu testen, werden, gemäß Abbildung 5.6, drei Phasen benötigt:

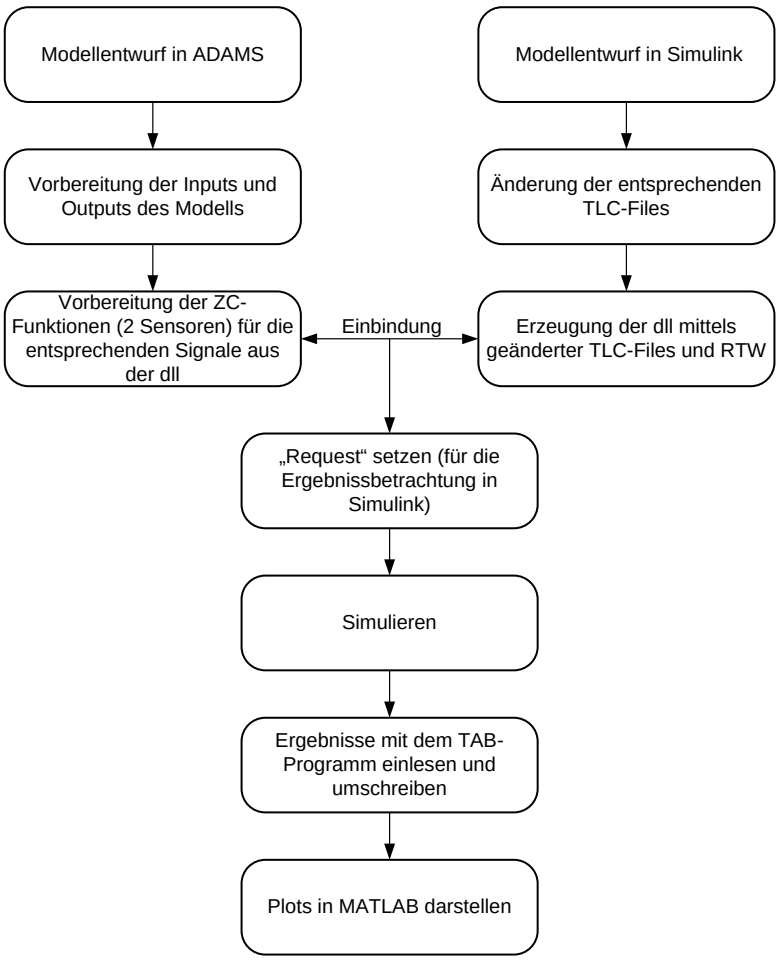


Bild 5.6: Simulationsablauf einer integrierten Simulation mit der ZC-HaSh-Methode

- **Vorbereitung des Simulink-Modells:**
  Das Modell wird hier in Blockschaltbildern entworfen. Anschließend werden die TLC-Files gemäß Abbildung 5.5 verändert. Somit können alle ZC-Signale in Form von zusätzlichen Ausgangsvariablen in der Modellbeschreibung hinzugefügt werden. Die

Kompilierung des Modells mittels des *Target Language Compilers* liefert dann die gewünschte DLL-Datei.

- **Vorbereitung des ADAMS-Modells:**
  Das Teilmodell wird in ADAMS entworfen. Eingänge und Ausgänge des Modells für die spätere Anbindung der GSE werden definiert. Um den Verlauf einer Variablen in Abhängigkeit von den Integrationsschritten in MATLAB darzustellen, wird ein *Request*-File gesetzt, das mit dem TAB-Programm eingelesen wird. Für jedes ZC-Signal aus dem zu verbindenden Simulink-Modell wird eine ZC-Überprüfung (beschrieben im Unterkapitel 5.3.4) mit jeweils zwei Sensoren vorbereitet.

- **Kopplung der Teilmodelle und Simulation:**
  Nach den Vorbereitungen wird die in der ersten Phase erstellte DLL-Datei ins ADAMS-Modell geladen. Wichtig dabei ist die Übereinstimmung der Eingangs- bzw. Ausgangsvariablen des Simulink-Modells mit den Ausgangs- bzw. Eingangsvariablen des ADAMS-Modells. Nach der Simulation können die Daten aus dem *Request*-File mit dem TAB-Programm eingelesen und in einem MAT-File gespeichert werden. Dieses File kann im Workspace von MATLAB geladen werden. Die Zeichnung der Variablen mit *Markern* zeigt die genauen Integrationspunkte während der Simulation.

6 Umsetzung und Test des Konzepts

Um das in Kapitel 5 vorgestellte Konzept zu testen, werden zunächst einige Beispielmodelle verwendet. Die Modelle werden immer in zwei Teile getrennt, so dass ein Teil in Simulink und das andere in ADAMS erstellt wird. Im Laufe dieses Kapitels werden drei Beispiele mit steigender Komplexität beschrieben. Die Ergebnisse der Simulationen mit verschiedenen Integrationsschrittweiten bei teilweisem Einsatz der ZC-HaSh-Technik werden miteinander verglichen. Aussagen bezüglich Simulationsrechenzeit, Speicher und Effizienz der Ergebnisse können dann getroffen werden.

6.1 Fremderregtes Feder-Masse-System

6.1.1 Beschreibung

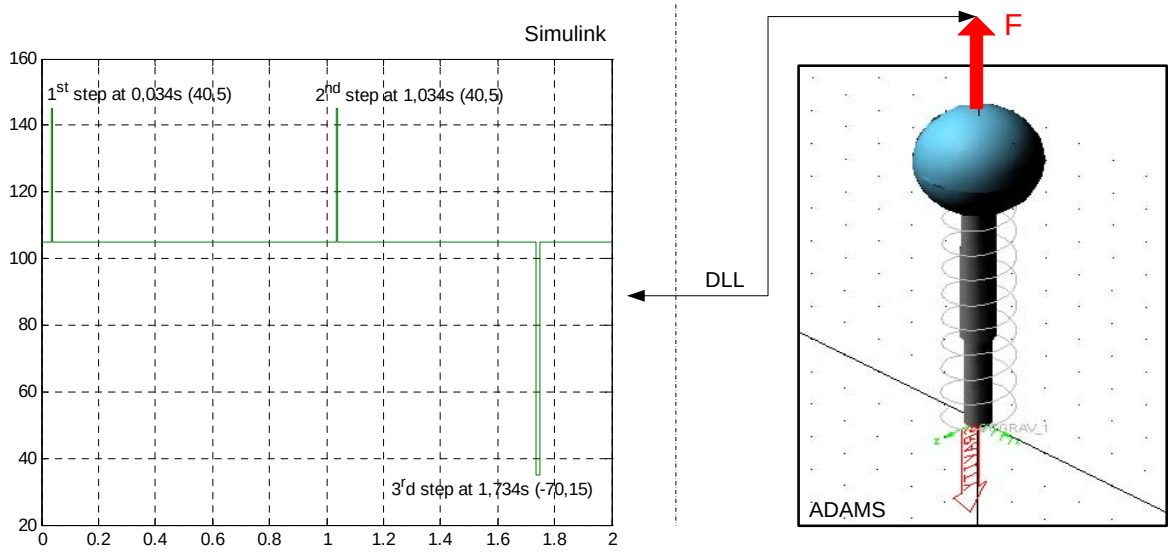


Bild 6.1: Ein Feder-Masse-Modell mit einer Fremderregung

Es handelt sich in der Abbildung 6.1 um ein Feder-Masse-System, das mit einer Kraft angeregt wird. Die Masse hat dabei den Wert $m = 10kg$. Der Feder besitzt die Steifigkeit $c = 46N/m$ und die Dämpfung $D = 12N \cdot s/m$. Die Gravitationskraft hat den Wert $g = 9.81kg/m^2$. Die anregende Kraft bekommt ihre Werte aus einer DLL-Datei, die in Simulink erstellt wird. Die Kraft hat einen konstanten Wert von $105N$ mit einer negativen und zwei positiven Step-Funktionen an den Simulationszeiten $0.034s$, $1.034s$ und $1.734s$. Die Breite und die Amplitude der überlagerten Funktionen sind den Vektoren $(0.005s; 40N)$, $(0.005s; 40N)$ und $(0.015s; 70N)$ zu entnehmen.

6.1.2 Simulation ohne Sensoren

Um die Situation ohne die Verwendung der Sensoren zu analysieren, wird das Modell in verschiedenen Konfigurationen simuliert:

- Das Modell wird komplett in ADAMS entworfen und mit einer sehr kleinen Schrittweite[1] simuliert.
- Das Modell wird komplett in Simulink entworfen (Abbildung 6.2) und mit einer engen Integrationstoleranz simuliert. Es wird ein Integrator mit einer variablen Schrittweite benutzt, bei der der größte Schritt $0.001s$ beträgt und der Toleranzfaktor ein Wert von 10^{-8} hat.

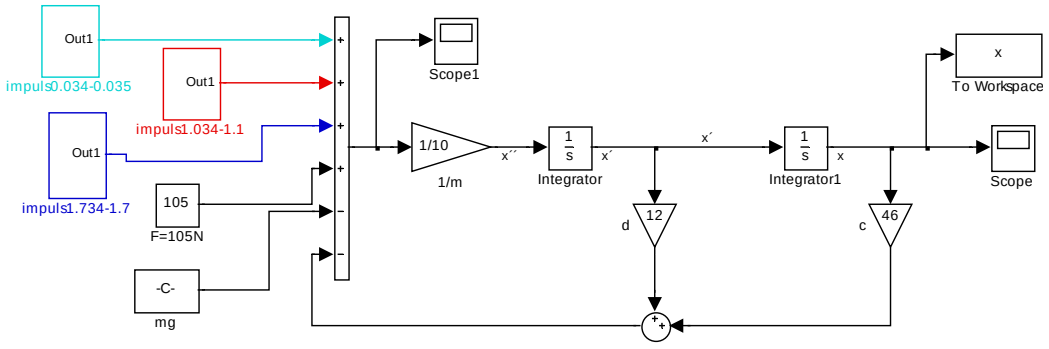


Bild 6.2: Ein Feder-Masse-Modell in Simulink

- Ein Kombimodell: das mechanische Teil wird in ADAMS implementiert und die Kraft in einem Simulink-Modell mit einer DLL erstellt. Das Modell mit der gebundenen GSE wird mit verschiedenen Schrittweiten simuliert

[1] Für die Festlegung der Schrittweite in ADAMS wird nur die Möglichkeit zur Eingabe der Ausgabeschrittweite gegeben. Sie steht aber in einem engen Zusammenhang mit der Integrationsschrittweite.

Die beiden ersten Konstellationen sind als Referenzkurven anzusehen. Da sie mit kleinen Integrationsschrittweiten und „harten“ Integrationskriterien simuliert wurden, liefern sie genaue Ergebnisse.

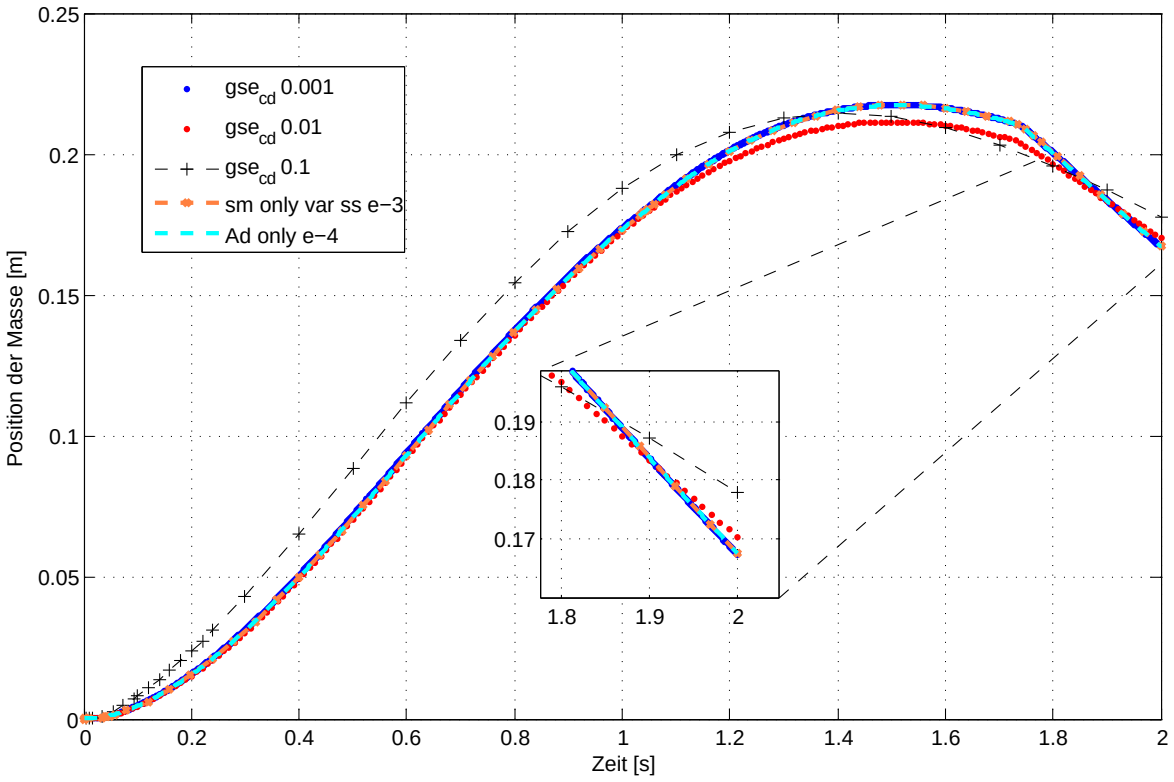


Bild 6.3: Ergebnisse der Simulationen ohne Sensoren

Bei der dritten Konstellation wird die GSE verwendet. Abbildung 6.3 zeigt: je kleiner die Schrittweite ist, desto genauer werden die Ergebnisse. Eine Simulation mit der Integrationsschrittweite $\delta t = 0.001$ liefert zwar genaue Ergebnisse, ist aber zeit- und speicheraufwendig.

6.1.3 Simulation mit Sensoren

Mit der Verwendung von Sensoren werden die drei *Step*-Funktionen aus der DLL sehr genau aufgelöst bei Erhalt einer globalen Schrittweite von $\delta t = 0.1$. Abbildung 6.4 zeigt: während die Simulation mit der globalen Schrittweite $\delta t = 0.1$ ohne Sensoren von den Referenzkurven sehr abweicht, weist die Simulation mit den gleichen Bedingungen unter Verwendung von Sensoren die gleiche Ergebnisgenauigkeit wie die Referenzkurven auf.

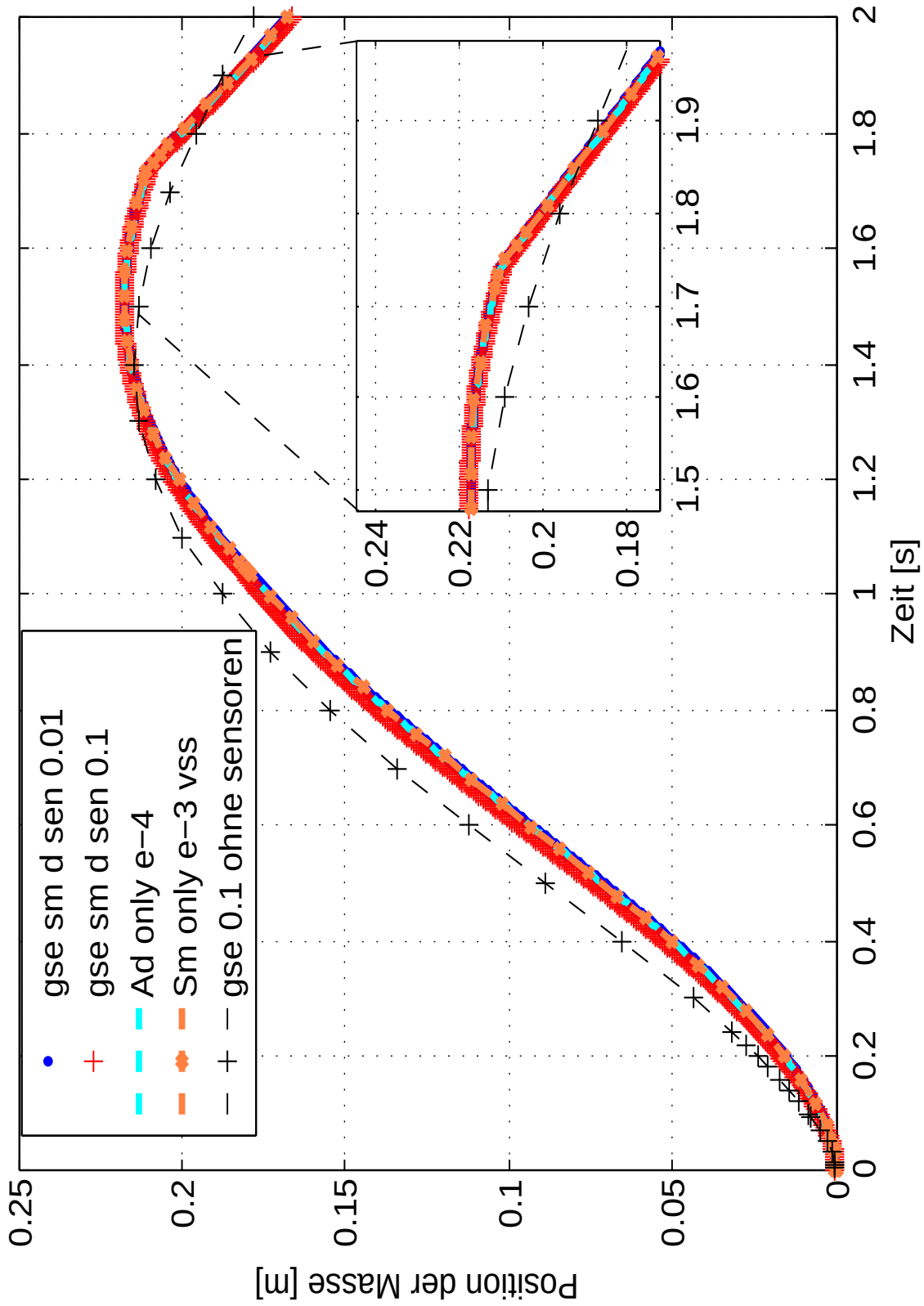


Bild 6.4: Ergebnisse der Simulationen mit Sensoren

6.2 Regelung eines rotierenden inversen Pendels (RODEL)

RODEL ist ein im Institut für Regelungstechnik an der TU-Braunschweig entstandener Versuchsaufbau. Es geht um einen mechanischen Aufbau eines in sich instabilen Systems, das in einer metastabilen Lage durch einen Zustandsregler stabilisiert wird. Um die ZC-HaSh-Methode zu testen, wird der mechanische Aufbau des RODEL in ADAMS nachgebildet und der Regler aus früheren Arbeiten in MATLAB/Simulink übernommen. Zum Vergleich wird eine gekoppelte Simulation über die GSE-Schnittstelle mit und ohne Verwendung der ZC-HaSh durchgeführt.

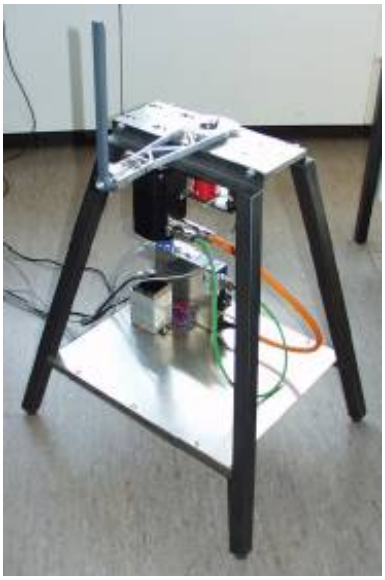


Bild 6.5: Der reale Versuchsaufbau: RODEL

6.2.1 Beschreibung

Das rotierende, inverse Pendel besteht aus folgenden Teilen:

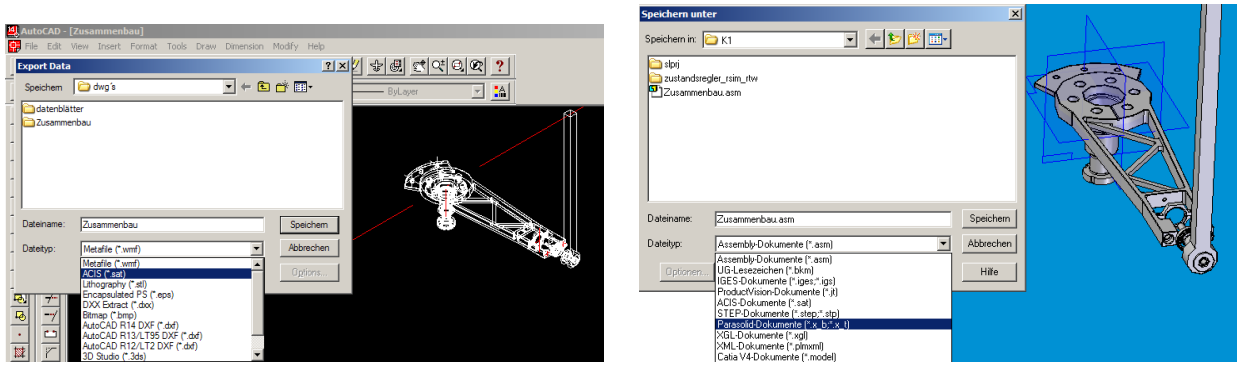
- Einem Trägerarm, welcher sich in der horizontalen Ebene drehen kann. Angetrieben wird er über einen Riemen von einem Motor, welcher vertikal darunter angeordnet ist. Der Trägerarm kann beliebig oft im oder gegen den Uhrzeigersinn ohne mechanische Einschränkungen rotieren.
- Eine Pendelstange am Ende einer Seite des Trägerarms, welche um 360° in einer Ebene senkrecht zum Trägerarm rotieren kann.

Die Position des Trägerarms wird durch einen im Motor angebrachten Inkrementalgeber gemessen. Am Pendel selbst ist ein Potentiometer angebracht, welches die Position des Pendels in der vertikalen Ebene bestimmt.
Der Trägerarm, die Pendelstange, die Grundplatte, sowie andere Verbindungselemente lagen aus früheren Arbeiten im DWG-Format vor, einem eigenständigen Format von *AutoCad*.
DWG[2] ist ein Grafikformat für 2D- (bzw. 3D-) Vektorgrafiken. Eine *DWG* enthält nur Vektoren- und Endpunkteinformationen. ADAMS unterstützt unter anderem auch dieses Format.
Mit dem Import solch eines Formates in ADAMS wird jedoch nur die Geometrie des Modells wiedergegeben. Zudem kann das importierte Modell nur als Teil eines schon vorhandenen Modells eingefügt werden. Dieses Problem wird umgangen, indem man das importierte Modell zum *Ground*[3] hinzufügt. Das erhaltene Modell besteht nicht aus Volumenkörpern, ist dementsprechend auch masselos, hat somit keine Trägheit und es lassen sich keine Parameteränderungen durchführen.

Konvertierung vom CAD-Format(*.DWG) in das ACIS-Format(*.sat)

Um die Simulationen mit ADAMS durchführen zu können, muss die Zeichnung ($*.dwg$) zu einem Volumenkörper konvertiert werden.
Für die Geometriebeschreibung, Parameteränderung und Darstellung von $3D$-Volumenkörpermodellen benutzt *AutoCad* den Volumenkernel $ACIS$[4]. $ACIS$ ist in der objektorientierten Programmiersprache C++ geschrieben, und ist mittlerweile zu einem Quasi-Standard bei CAD-Programmen geworden. Dateien im $ACIS$-Format tragen üblicherweise die Dateinamen-Erweiterung ($*.sat$) [68].
$ACIS$-Format wird allerdings von *Adams* nicht unterstützt. Deswegen wird die *AutoCad*-Erweiterung *MechanicalDesktop* für die mechanischen 3D-Konstruktionen verwendet, um das Modell in das *ACIS*-Format konvertieren zu können. Die Abbildung 6.6(a) zeigt die Konvertierungsumgebung in *MechanicalDesktop*.

[2] (∗.dwg) steht für [engl.] *Drawing* (Zeichnung)
[3] stellt das globale Koordinatensystem dar, welches den globalen Ursprung $(0, 0, 0)$ definiert. Bei der Konstruktion eines Modells wird das Ground von ADAMS als Teil automatisch erstellt
[4] steht für Alan, Charles, Ian's System



(a) von DWG zu ACIS (AutoCad) (b) von ACIS zu Parasolid (Solidworks)

Bild 6.6: Konvertierung des Modells als Vorbereitung für die Simulation in ADAMS

Konvertierung vom ACIS-Format ins PARASOLID-Format

Für die Geometriebeschreibung, Parameteränderung und Darstellung von $3D$-Volumenkörpermodellen benutzt ADAMS im Gegensatz zu *AutoCad* den Modellierkern[5]-*Parasolid*.
Die Aufgabe des *Parasolid* ist es, dem darauf aufsetzenden Programm (hier ADAMS) Funktionen zur Verfügung zu stellen, mit denen Volumenkörper mathematisch beschrieben werden können. Darüber hinaus werden Funktionen zur Verfügung gestellt, die das Verändern der Körper und ihre Darstellung auf dem Bildschirm ermöglichen [68].
Daraus resultiert die Notwendigkeit einer Schnittstellensoftware, die sowohl *ACIS* als auch *Parasolid* unterstützt bzw. *ACIS* in *Parasolid* konvertieren kann. *CAD*-Programme wie *Solidworks* und *Solid Edge* bieten sich hier an.
Im Rahmen dieser Arbeit wurde *Solid Edge V17* benutzt. Die Software erlaubt das Reparieren und Vernähen der freien Flächen beim Übersetzen von ACIS-Daten, so dass Volumenkörper erstellt werden können. So werden beispielsweise zwei Flächen an einer gemeinsamen Schnittkante miteinander verbunden. Umschließen die Flächen ein Volumen, so wird ein Volumenkörper erzeugt.
Die erhaltene ($*.xt$) Datei kann nun in ADAMS importiert werden.

6.2.2 Bearbeitung des Modells für die Simulation

Das in ADAMS importierte Modell besteht zunächst einmal aus 26 Teilen. Bauteile wie z.B. das Riemen-Zahnrad, die Passhülse und die Unterlegscheiben, wurden gelöscht, weil sie für die Simulation irrelevant sind. Die Messvorrichtung (Potentiometer), der Flansch, die

[5]Kern zur Geometriebeschreibung und -darstellung, der in 3D-CAD-Systemen verwendet wird

Unterlegscheiben sowie die Schrauben, die sich auf dem Trägerarm befinden, tragen dagegen zur Gesamtmasse des Trägerarms bei und beeinflussen die dynamischen Eigenschaften des Modells. Deshalb werden diese Teile zur Vereinfachung des Modells mit dem Trägerarm zu einem einzigen Körper zusammengefasst. Somit lassen sich Änderungen der Eigenschaften oder der Darstellung mit einem geringeren Aufwand durchführen.

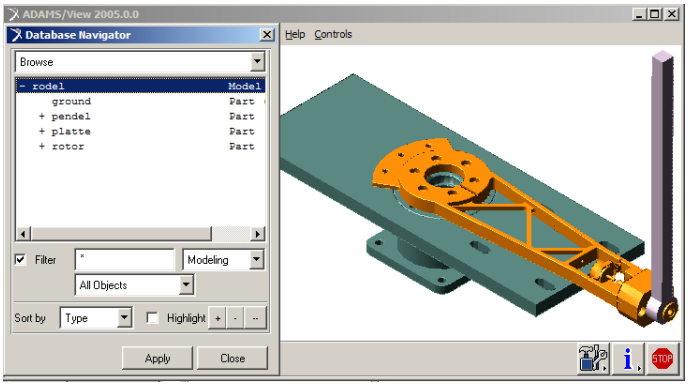


Bild 6.7: Vereinfachtes RODEL-Modell

Die Anzahl der Bauteile wurde also auf drei Hauptbauteile reduziert:

1. die Grundplatte,
2. der Trägerarm und
3. die Pendelstange

Abbildung 6.7 zeigt das vereinfachte Modell in ADAMS.

Definition der Massen

Die für eine Simulation, bzw. Kinematik- und Dynamikanalyse notwendigen physikalischen Eigenschaften des Modells[6] werden von ADAMS berechnet. Vor dem Berechnen der physikalischen Eigenschaften eines Teils müssen die Materialeigenschaften und mechanischen Eigenschaften festgelegt werden. Man kann entweder das Material angeben oder die Masse selbst definieren. ADAMS weist im ersten Fall dem Teil eine Masse zu.
Die Masse der Pendelstange beträgt $0,105kg$. Die Masse des Trägerarms, welcher die Messvorrichtung und Verbindungsteile enthält, beträgt $1,230kg$. Beide Massen wurden mit einer Waage bestimmt. Die Flanschmasse wurde auf $0,1kg$ geschätzt.

[6] Schwerpunkte der Teile, Massenträgheitsmomente usw...

Definition der Drehverbindungen (Revolute Joints)

Eine Drehverbindung verbindet zwei Körper und lässt dabei einen rotatorischen Freiheitsgrad offen. Eine Bewegung zwischen den definierten Körpern ist somit nur noch im offenen Freiheitsgrad möglich. [69]
Die Pendelstange kann in einer Ebene senkrecht zum Trägerarm um 2π rotieren. Letzterer kann in der horizontalen Ebene ebenfalls um 2π rotieren.
Zur Definition der Gelenke wird für jeden Körper ein Hilfskoordinatensystem benötigt. Es bestimmt die Lage der Drehachse und deren Orientierung. Die Marker der beiden Hilfskoordinatensysteme können später für eine Winkelmessung benutzt werden.

Konstruktion des Gestells

Das Gestell besteht aus einer Platte und vier Stangen. Für die Konstruktion der Stangen wird ein Zylinder zwischen zwei Designpunkten (Anfangs- und Endpunkt) konstruiert. Das „manuelle“ Erstellen bzw. Einsetzen von Designpunkten erfolgt in ADAMS nur mit Hilfe des Arbeitsgitters eines $2D$-Koordinatensystems. Für die Erstellung der Endpunkte wurde deshalb auf Designtabellen zurückgegriffen. Vorher wird das Arbeitsgitter auf die Höhe des gewünschten Einsatzortes der oberen Platte gebracht. Anschließend wird die Ebene ausgewählt, in der die obere Platte liegen soll. Auf dieser Ebene wird dann die Platte gezogen.

Erstellen von Zustandsvariablen

Das RODEL soll später mit einem Zustandsregler ausgeregelt werden. Die hier für den Ansatz eines Zustandsreglers benötigten Zustandsgrößen des Systems sind:

- Winkel des Trägerarms
- Winkelgeschwindigkeit des Trägerarms
- Winkel des Pendels
- Winkelgeschwindigkeit des Pendels

Die Zustandsgrößen des Systems sind hier alle direkt messbar. Die einzige Stellgröße des Systems ist das Moment des Antriebes, welches am Trägerarm angreift. Mit dem *Function-Builder* lassen sich Funktionen mit zeitveränderlichen Größen des Systems aufstellen (Abstände, Winkel, Geschwindigkeiten...).

Der aktuelle Winkel des Pendels ergibt sich beispielsweise aus der folgenden Formel:

```
MOD( AZ(.rodel.pendel.MARKER 6 , .rodel.rotor.MARKER 5 ) , 2*PI )
```

Die Funktion `'AZ(x1, x2)'` gibt den Winkel zwischen zwei Koordinatensystemen an, die sich relativ zueinander bezüglich der z-Achse drehen.
Hier werden Koordinatensysteme verwendet, die in Verbindung mit der Definiton des *Revolut Joint* erstellt worden sind. Zu beachten ist, dass die Drehachse (z-Achse) in diesem Fall nicht

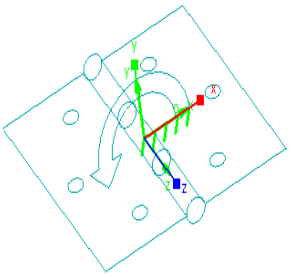


Bild 6.8: Hilfskoordinatensystem in der Drehverbindung

die globale z-Achse ist, sondern die relative z-Achse der beiden Hilfskoordinatensysteme. Die Drehachse hängt also immer von der Lage der Drehverbindung ab.
Mit der Funktion `'MOD(x1, x2)'` wird der aktuelle Winkel modulo 2π ($2 \quad \% \quad 2\pi$) berechnet. Die Winkelgeschwindigkeit des Pendels ergibt sich aus der folgenden Formel:

```
WZ(.rodel.pendel.MARKER 6 , .rodel.rotor.MARKER 5 )
```

wobei die Funktion `'WZ(x1, x2)'` die Differenz zwischen den Winkelgeschwindigkeiten von zwei Markern des Koordinatensystems angibt.
Analog ergeben sich für den Rotor die Funktionen für Winkel bzw Winkelgeschwindigkeit zu:

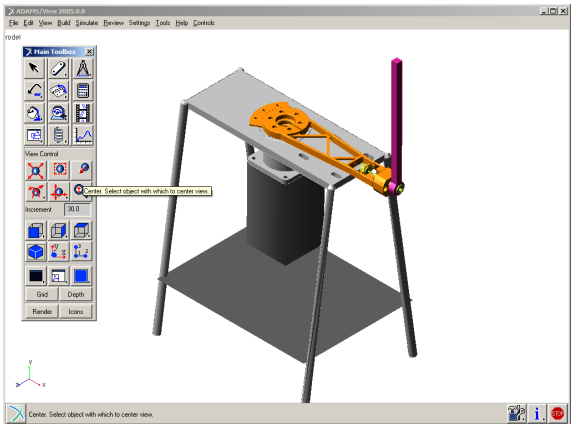
```
MOD( AZ(.rodel.pendel.MARKER 8 , .rodel.rotor.MARKER 711 ) ,2*PI )

WM(.rodel.rotor.MARKER 8 , .rodel.platte.MARKER 711 )
```

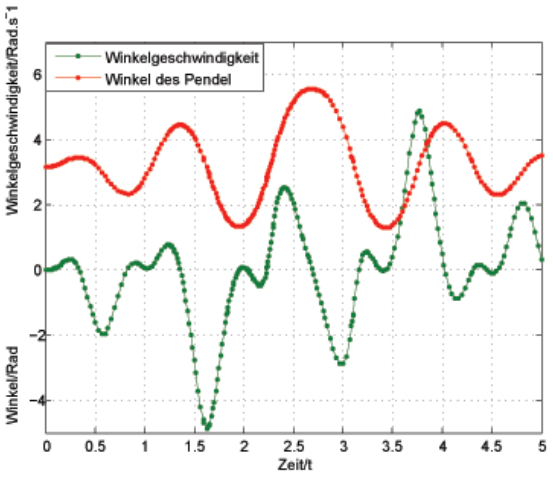
Für den Stelleingriff wird eine ADAMS-Zustandsvariable; `moment`, erstellt. Bei der Kopplung von ADAMS und MATLAB/Simulink wird der Wert dieser Zustandsvariablen in bestimmten Zeitabständen vom regelungstechnischen Programm aktualisiert.

6.2.3 Prototypentest

Abbildung 6.9(a) zeigt den Prototypen mit Pendelstange, Trägerarm und Gestell.



(a) RODEL-Prototyp in ADAMS



(b) Zustandsvariablenverläufe bei einem sinusförmigen Anregungsmoment M_{Test}

Bild 6.9: Test des RODEL-Prototypes

Für den Test des Prototypen wird ein Antriebsmoment erstellt, welches am Trägerarm ansetzt.
Abbildung 6.9(b) zeigt die Kurvenverläufe der Zustandsgrößen des Systems für ein sinusförmiges Antriebsmoment $M_{test} = 20 \cdot cos(2 \cdot \pi \cdot t)$.

6.2.4 Mathematische Beschreibung des RODELs

Anhand der Lagrange-Mechanik werden die Differenzialgleichungen zur Beschreibung der Bewegungen von Pendelstange und vom Trägerarm aufgestellt:

- Zuerst werden die *Zwangsbedingungen* bestimmt, woraus sich die Anzahl der generalisierten Koordinaten ergibt. Es können vier Zwangsbedingungen ausgemacht werden. Zum einen ist die Länge des Arms und des Pendels konstant. Zum anderen können sich sowohl der Arm als auch das Pendel jeweils nur in einer Ebene bewegen. Benötigt werden also:

  $$f = 3N - k = 3 \cdot 2 - 4 = 2$$

  generalisierte Koordinaten. Als generalisierte Koordinaten werden der Winkel des Armes q_1 und der Winkel des Pendels q_2 bezogen auf die obere, instabile Ruhelage gewählt.

- Betrachtet wird die Bewegung des Pendelschwerpunktes.
  Der Ortsvektor im kartesischen Koordinatensystem wird in die generalisierten Koordinaten q_1 und q_2 transformiert:

$$\begin{pmatrix} x_{Pendel} \\ y_{Pendel} \\ z_{Pendel} \end{pmatrix} = \begin{pmatrix} l_1 . \cos q_1 - l_2 . \sin q_2 . \sin q_1 \\ l_1 . \sin q_1 + l_2 . \sin q_2 . \cos q_1 \\ l_2 . \cos q_2 \end{pmatrix}$$

- Die kinetische Energie T und potentielle Energie V des System werden als Funktionen von $\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}$ angegeben:

$$\begin{aligned} T_{Arm} &= \tfrac{1}{2} \cdot J_{Arm} \cdot \dot{q}_1^2 \\ T_{Pendel} &= 1/2 \cdot m_p \cdot v_{Pendel}^T \cdot v_{Pendel} + 1/2 \cdot J_{Pendel} \cdot \dot{q}_2^2 \\ V_{Arm} &= 0 \\ V_{Pendel} &= m_p \cdot g \cdot l_2 \cdot \cos q_2 \end{aligned}$$

  Mit der Anwendung des Ansatzes von Lagrange

$$L = T_{Arm} + T_{Pendel} - V_{Pendel}$$

  ergibt sich nach einigen Umformungen für die Lagrangefunktion:

$$\begin{aligned} L = \tfrac{1}{2} m_p \cdot l_1^2 \cdot \dot{q}_1^2 + \tfrac{1}{2} m_p \cdot l_2^2 \cdot \dot{q}_2^2 + m_p \cdot l_1 \cdot l_2 \cdot \dot{q}_1 \cdot \dot{q}_2 \cos q_2 - \tfrac{1}{2} m_p \cdot l_2^2 \cdot \dot{q}_1^2 \cdot \sin^2 q_2 \\ + \tfrac{1}{2} J_{Arm} \cdot \dot{q}_1^2 + \tfrac{1}{2} J_{Pendel} \cdot \dot{q}_2^2 - m_p \cdot g \cdot l_2 \cdot \cos q_2 \end{aligned}$$

- Die Bewegungsgleichung des Systems lässt sich aus den Lagrangegleichungen herleiten. Mit den generalisierten Kräften (deren Richtung mit den generalisierten Koordinaten übereinstimmt) ergibt sich die Lagrangegleichung zweiter Art zu:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial q_n}\right) = \frac{\partial L}{\partial q_n} + Q_n \qquad wobei \qquad Q_n = \begin{pmatrix} M_a - M_{r1} \\ M_{r2} \end{pmatrix}$$

  M_a ist das von außen angreifende Moment, M_{ri} sind die Reibmomente im Motor und im Pendellager.

Die nichtlineare Bewegungsgleichung ergibt sich dann wie folgt:

$$\begin{pmatrix} M_a \\ 0 \end{pmatrix} = \begin{pmatrix} p_1 + p_2 \cdot \sin^2 q_2 & p_3 \cdot \cos q_2 \\ p_3 . \cos q_2 & p_4 \end{pmatrix} \cdot \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix}$$

$$+ \begin{pmatrix} C_1 + p_2 \cdot \dot{q}_2 \cdot \sin 2q_2 & -p_3 \cdot \dot{q}_2 \cdot \sin q_2 \\ -\frac{1}{2} p_2 \cdot \dot{q}_1 \cdot \sin 2q_2 & C_2 \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

$$+ \begin{pmatrix} 0 \\ -p_5 \cdot \sin q_2 \end{pmatrix}$$

wobei

$$\begin{aligned} p_1 &= m_p \cdot l_1^2 + J_{Arm} \\ p_2 &= m_p \cdot l_2^2 \\ p_3 &= m_p \cdot l_1 \cdot l_2 \\ p_4 &= m_p \cdot l_2^2 + J_{Pendel} \\ p_5 &= m_p \cdot l_2 \cdot g \end{aligned}$$

Das System wird mit der Potenzreihenentwicklung von Taylor um den Arbeitspunkt $q_{2,0} = \dot{q}_{1,0} = \dot{q}_{1,0}$ linearisiert. Die Entwicklung wird dabei nach dem linearen Glied abgebrochen.
Für eine Funktion mit zwei Veränderlichen $f = f(x_1, x_2)$ gilt als lineare Nährung um $\mathbf{x}_0 = (x_{1,0}; x_{2,0})$ das Taylor-Polynom ersten Grades:

$$T_1(\mathbf{x}; \mathbf{x}_0) = f(\mathbf{x}_0) + (x_1 - x_{1,0}) \frac{\partial f(\mathbf{x}_0)}{\partial x_1} + (x_2 - x_{2,0}) \frac{\partial f(\mathbf{x}_0)}{\partial x_2}$$

Das linearisierte System hat dann die Form:

$$\begin{pmatrix} M_a \\ 0 \end{pmatrix} = \begin{pmatrix} p_1 & p_3 \\ p_3 & p_4 \end{pmatrix} \cdot \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} + \begin{pmatrix} C_1 & 0 \\ 0 & C_2 \end{pmatrix} \cdot \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -p_5 \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$$

Ausführliche Zwischenrechnungen sind verschiedenen Studien- und Diplomarbeiten vom Institut für Regelungstechnik Braunschweig zu entnehmen.

Als Ergebnis der Modellbildung und Linearisierung liegen die Bewegungsgleichungen des Systems in Zustandsraumdarstellung vor:

$$\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{p_3 p_5}{p_1 p_4 - p_3^2} & -\frac{p_4 C_1}{p_1 p_4 - p_3^2} & \frac{p_3 C_2}{p_1 p_4 - p_3^2} \\ 0 & \frac{p_1 p_5}{p_1 p_4 - p_3^2} & \frac{p_3 C_1}{p_1 p_4 - p_3^2} & -\frac{p_1 C_2}{p_1 p_4 - p_3^2} \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 2.M_0.\frac{p_4}{p_1 p_4 - p_3^2} \\ 2.M_0.\frac{p_3}{p_1 p_4 - p_3^2} \end{pmatrix} .u$$

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

Diese Gleichungen bilden die Grundlage für den Aufbau des Modells in Simulink und für den weiteren Reglerentwurf.

6.2.5 Reglerauslegung

In Rahmen dieser Arbeit soll als Regler eine Zustandsrückführung zum Einsatz kommen.Grundidee eines Zustandsregler ist es, die Zustände -also die inneren Informationen aus dem Prozess- rückzukoppeln. Dies steht im Gegensatz zum Standardregelkreis, wo nur die Regelgröße -also der Ausgang des Prozess- rückgekoppelt wird. Vorraussetzung für die Anwendung eines Zustandsregler ist die Steuerbarkeit des zu regelnden Systems.
Ein nicht sprungfähiges Mehrgrößensystem lässt sich wie folgt beschreiben:

$$x = \underline{\underline{A}}\ \underline{x} + \underline{\underline{B}}\ \underline{u}$$

$$y = \underline{\underline{C}}\ \underline{x}$$

Unter der Annahme, dass die Zustandsgrößen $\underline{x}$ wiederum vollständig messbar sind, kann mit einer Rückführungsmatrix $\underline{\underline{R}}$ der Zustand auf den Stellgrößenvektor zurückgeführt werden. Durch einer Matrix $\underline{\underline{V}}$ wird die Führungsgenauigkeit sichergestellt. Das Zustandsgeregelte System wird durch folgende Gleichungen beschrieben:

$$x = (\underline{\underline{A}} + \underline{\underline{B}}\ \underline{\underline{R}})\ \underline{x} + \underline{\underline{B}}\ \underline{\underline{V}}\ \underline{w}$$

$$y = \underline{\underline{C}}\ \underline{x}$$

Für den Entwurf eines Zustandsregler, also die systematische Bestimmung von $\underline{\underline{R}}$ gibt es

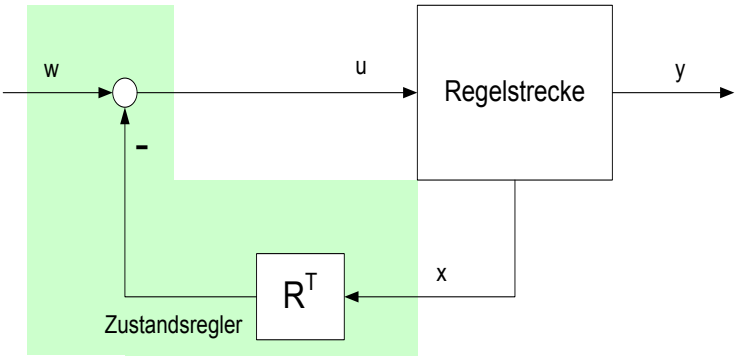


Bild 6.10: Zustandsregler

zwei Hauptmethoden:

- Polvorgabe: Es stehen n Regelparameter zur Verfügung, mit denen es möglich ist, die n Pole des charakteristischen Polynoms an gewünschte Stellen zu verschieben.
- Optimierung: Die Minimierung einer Quadratischen Verlustfunktion führt zu einer global optimalen einfachen Lösung.

$$J = \int_0^\infty \left(\underline{x}^T \underline{\underline{W_x}} \underline{x} + \underline{u}^T \underline{\underline{W_u}} \underline{u} \right) dt$$

Mittels eines dieser Verfahren wird ein geeigneter Regelvektor R bestimmt, dieser bestimmt die Stellgröße $u(t)$ durch Rückführung der Zustandsgrößen x des Systems:

$$\underline{u}(t) = -\underline{R} \cdot \underline{x}(t), \qquad \underline{R} = (r_1 \quad r_2 \quad r_3 \quad r_4)^T \tag{6.1}$$

Nähere Grundlagen zum Zustandsregler können z.B. in [71] und in [72] nachgelesen werden. In Abbildung 6.11 zeigt die Struktur des Zustandsreglers des Rodels in Simulink. Als Ein- und Ausgänge der Strecke bzw. des Reglers werden folgende Größen benötigt:

- Winkel des Trägerarms
- Winkelgeschwindigkeit des Trägerarms
- Winkel des Pendels
- Winkelgeschwindigkeit des Pendels
- Antriebsmoment

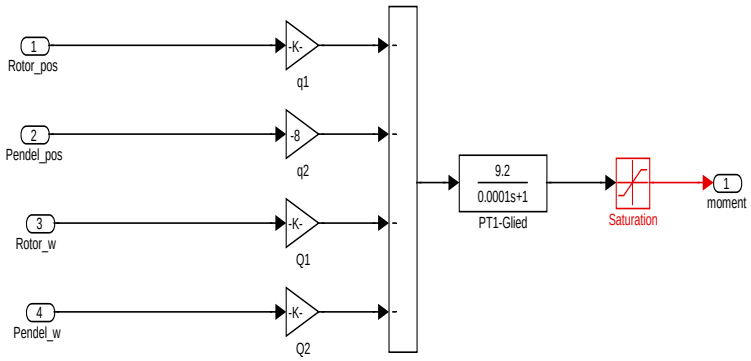


Bild 6.11: Zustandsregler in Simulink

6.2.6 Ergebnisse

Um die ZC-HaSh-Methode zu testen wird ein Regelvorgang des Pendels von der Ruhelage mit und ohne diese Technik simuliert. Dabei wird der *„Saturation“*-Block für die Implementierung nach der in Kapitel 5 beschriebene Methode verwendet. Die Simulation mit der ZC-HaSh-Methode liefert bei einer eingestellten Schrittweite von $0.1s$ einen deutlich besseren Kurvenverlauf als ohne.
Beide Kurvenverläufe werden miteinander verglichen. Als Referenz dienen die Ergebnisse aus der Simulation mit einer Schrittweite von $0.001s$.
Wie auf der Abbildung 6.12 zu sehen ist, liegen die Kurven fast für die gesamte Simulation-

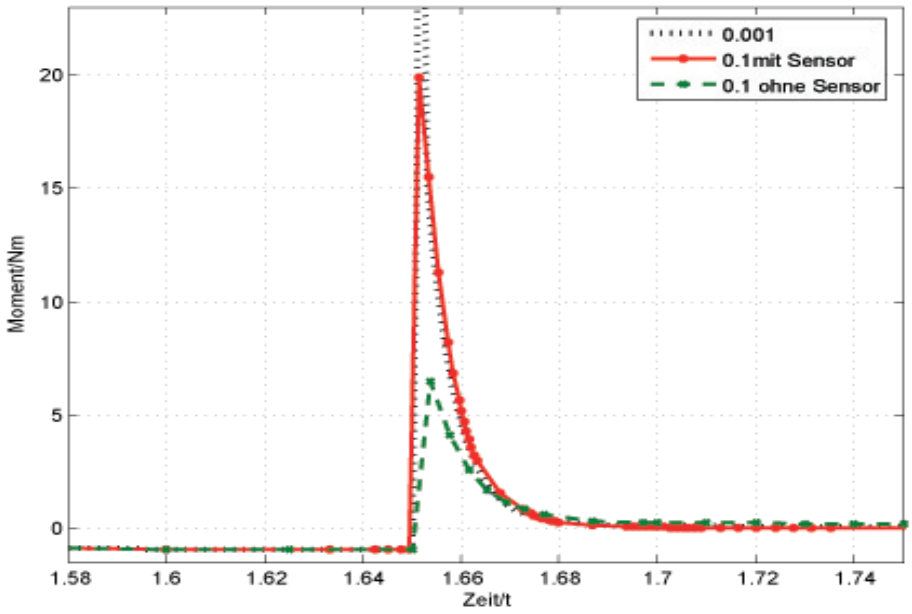


Bild 6.12: Vergleich der Momentenverläufe mit und ohne ZC-HaSh

zeit übereinander; es wurden die in der Tabelle 6.1 eingetragene Rechenzeiten, Iterationen und belegte Speicher gemessen.

Tabelle 6.1: Vergleich der Simulation mit und ohne ZC-HaSh

	Integrationsschritt	Rechenzeit	Iterationen	Speicherbedarf
ohne ZC-HaSh	0.1	3s	200	488KB
mit ZC-HaSh	0.1	3s	380	704KB
Referenz	0.001	146s	5020	9029KB

Insgesamt liefert die Simulation mit der ZC-HaSh-Methode hervorragende Ergebnisse. Aus der Tabelle 6.1 sind die großen Unterschiede im Rechen-, Zeit- und Speicheraufwand bei einem ähnlichen Kurvenverlauf zu erkennen. So dauert die Simulation bei einer Schrittweite von $0.001s$ knapp 49 mal länger, und braucht 13 mal soviel Speicher wie die Simulation mit der ZC-HaSh-Methode.
Mit diesen Einsparungen von Rechen-, Zeit- und Speicheraufwand können beim Entwicklungsprozess komplexer technischer Systeme Entwicklungszeiten deutlich verkürzt werden.

6.3 Gesamtes Fahrzeugmodell mit einem ABS

In diesem Beispiel wird ein MKS-Modell eines Fahrzeugs in ADAMS verwendet, während in MATLAB/Simulink ein ABS-Modell implementiert und später mittels der GSE in die gesamte Fahrzeugsimulation eingebunden wird. Das Potential der ZC-HaSh-Methode zur Verbesserung solcher Simulationen in Bezug auf ihre physikalische Plausibilität und gleichzeitig ihre Schnelligkeit wird hier überprüft und anschließend beurteilt.

6.3.1 Modellierung des ABS-Modells mit einem einfachen Fahrzeugmodell in MATLAB/Simulink

Automodell in Simulink

Ein Radmodell ist in Abbildung 6.13 dargestellt. Das Buch *Modellbildung und Simulation dynamischer Systeme* von Sherf [19] gibt eine genaue Beschreibung des Modells: das Rad rollt in die rechte Richtung mit der Winkelgeschwindigkeit ω_R. Der Bremsmoment M_B reagiert

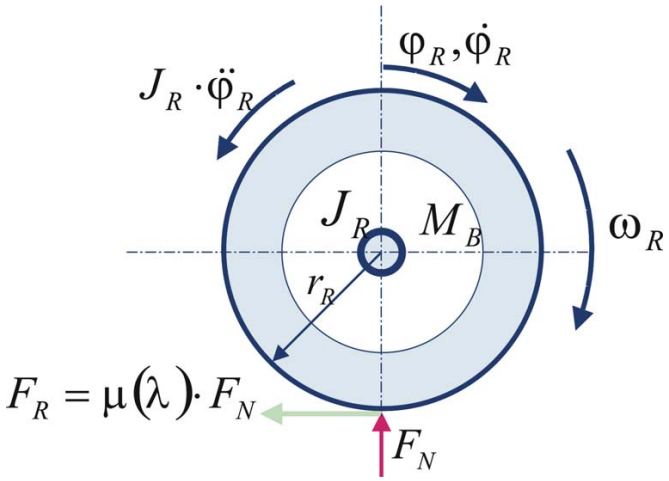


Bild 6.13: Radmodell

in die linke Richtung. Mit der Reibungskraft F_R schreibt sich die Momentengleichung wie folgt:

$$J_R \cdot \ddot{\alpha}_R = F_R \cdot r_R - M_B$$

Die Reibungskraft wird mittels eines Reibungskoeffizienten geschrieben: $F_R = -\mu(\lambda) \cdot F_N$.

Für ein trockenen Asphalt schreibt sich der Reibkoeffizient:

$$\mu(\lambda) = c_1 \cdot (1 - e^{-c_2 \cdot \lambda}) - c_3 \cdot \lambda$$

Der Schlupf am Rad wird mit der Hilfe der Radgeschwindigkeit v_R und Fahrzeuggeschwindigkeit v_F wie folgt definiert:

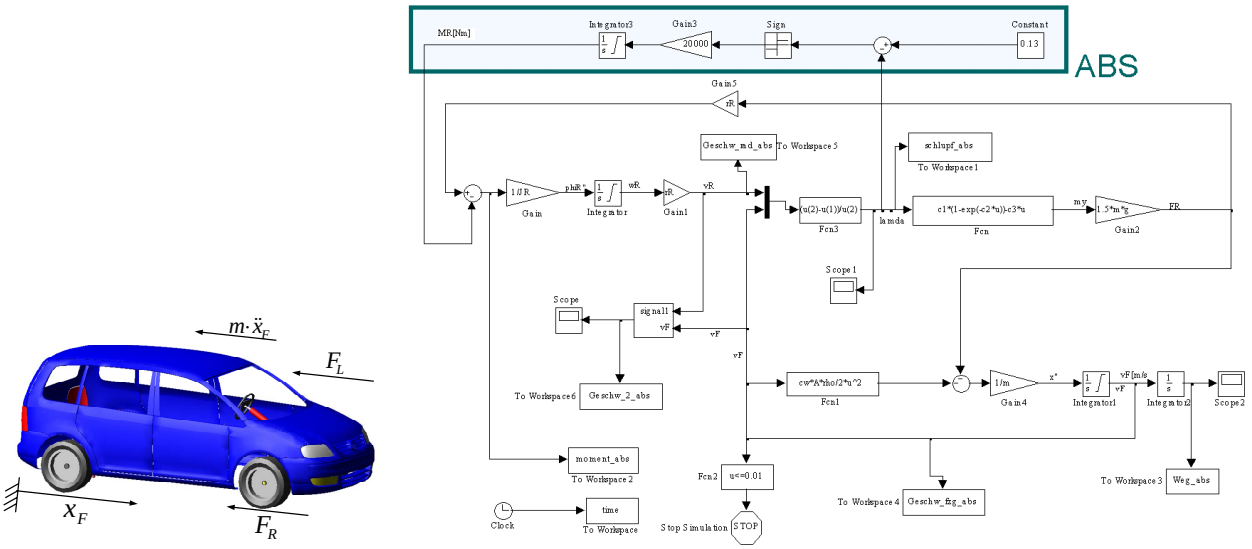
$$\lambda = \frac{v_F - v_R}{v_F}$$

In der Abbildung 6.14(a) werden die auf das Fahrzeug wirkenden Kräfte abgebildet. Man erhält die Gleichung:

$$m \cdot \ddot{x}_F = -F_R - F_L \tag{6.2}$$

$$= -\mu(\lambda) \cdot F_N - c_W \cdot A \cdot \frac{\rho}{2} \cdot \nu_F^2 \tag{6.3}$$

Aus den oben beschriebenen Gleichungen kann ein Simulink-Modell aufgestellt werden. In einem weiteren Schritt wird das Modell, wie in der Abbildung 6.14(b) dargestellt, um ein



(a) Fahrzegmodell (b) Simulink-Fahrzeugmodell mit einem ABS

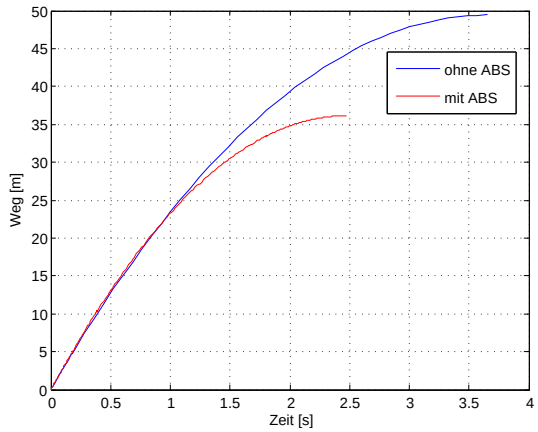
Bild 6.14: Modellierung eines Fahrzeuges in Simulink nach [19]

ABS-Submodell erweitert. Das ABS-Modell erhöht und vermindert das Bremsmoment je nach Schlupf. Der Schlupfwert bei einem maximalen Reibkoeffizienten beträgt 0.13.

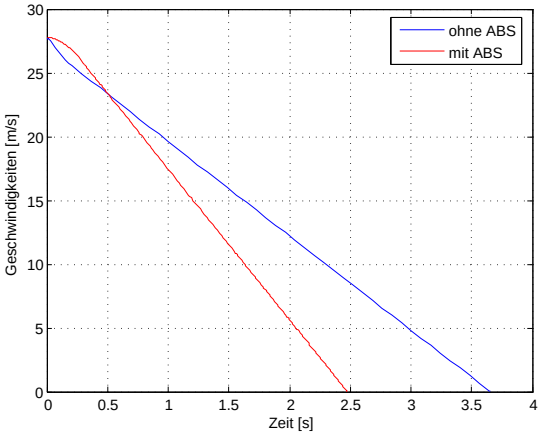
Funktionstest des Simulink-Modells

Für ein besseres Verständnis und eine Prüfung des ABS-Systems werden nach der Simulation in Simulink die Kurvenverläufe des Weges (Bild 6.15(a)), der Geschwindigkeit (Bild 6.15(b)), der Momente (Bild 6.15(c)) und der Schlupfwerte (Bild 6.15(d)) in der Abbildung 6.15 dargestellt.

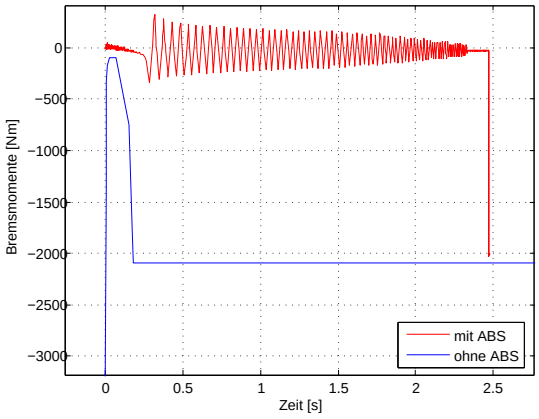
Bild 6.15(a) zeigt, dass der Bremsweg durch das ABS-System kürzer geworden ist ($37m$ statt $40m$ ohne ABS). Das Bild 6.15(b) zeigt, dass bei einer gleichen Anfangsgeschwindigkeit $v_0 = 27m/s$ das Fahrzeug mit ABS $1.1s$ schneller zum Stillstand ($v = 0m/s$) kommt als das Fahrzeug ohne ABS. Der Bremsmomentenverlauf in Bild 6.15(c) zeigt eine deutlich niedrigere Amplitude beim Fahrzeug mit ABS; das Signal ist aber sehr hochfrequent. In Bild 6.15(d) ist deutlich zu erkennen, dass der Schlupf beim Fahrzeug mit ABS den niedrigen Wert von 0.13 über den gesamten Bremsvorgang hat. Der Schlupf beim Fahrzeug ohne ABS hat dagegen den Wert 1, welcher als „Rutschen“ des Fahrzeugs zu interpretieren ist. Dies bewirkt einen verlängerten Bremsweg.



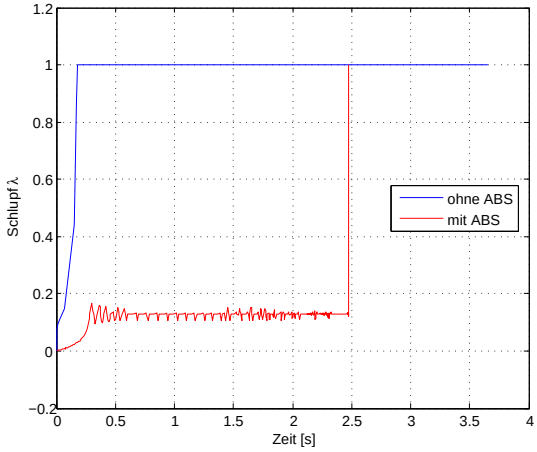
(a) Zurückgelegte Strecke bis zum Stillstand



(b) Geschwindigkeit



(c) Bremsmoment



(d) Schlupf

Bild 6.15: Test des Simulink-Modells

6.3.2 Vorbereitung des Fahrzeugsmodells für die gekoppelte Simulation

Bei einem vorhandenen Fahrzeugmodell in ADAMS wird das in Abbildung 6.16 dargestellte Bremssystem:
`mdids://acar_shared/templates.tbl/_brake_system_4Wdisk.tpl`
so verändert, dass dieses Subsystem direkt mit dem ABS-System aus der GSE kommuniziert und die in der DLL berechneten Bremsmomente einliest. Die Erstellung des neuen Bremssystems `_ABS_pre.tpl` wird in den folgenden Schritten beschrieben:

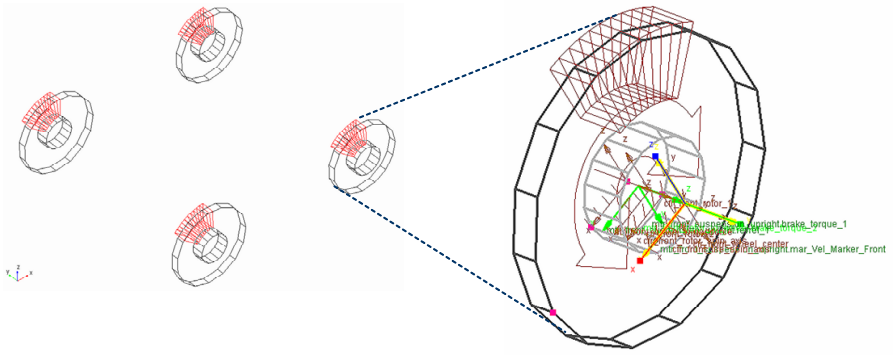


Bild 6.16: Bremssystem `_ABS_pre.sub`

Einfügen neuer Marker

Um die Geschwindigkeit des Fahrzeugs am Mittelpunkt jedes Rades zu bestimmen, werden für jedes Rad jeweils zwei symmetrische *Marker* eingebaut. Diese *Marker* werden wie folgt benannt:

- `mal_Vel_Marker_Front`
- `mar_Vel_Marker_Front`
- `mal_Vel_Marker_Rear`
- `mar_Vel_Marker_Rear`

Einfügen neuer Zustandsvariablen

Es werden insgesamt 16 neue Variablen generiert, die in 4 Gruppen aufgeteilt werden:

1. Eingänge des Systems:
   Hier werden die Werte für die jeweiligen Bremsmomente an den *vier* Rädern aus dem ABS-System hineingeschrieben:
   - `From_ABS_Left_Front`
   - `From_ABS_right_front`
   - `From_ABS_right_rear`
   - `From_ABS_Left_Rear`

2. Variablen zur Bestimmung der relativen Radgeschwindigkeiten:
   Sie setzen sich aus der Umrechnung der Winkelgeschwindigkeiten an den Rädern zusammen. Die jeweiligen Funktionen werden unter jeder Variable explizit angegeben:

   - `left_front_brake_velocity`
     Funktion:
     `(PI/180)*326*WZ(._ABS_pre.mtl_front_rotor_to_wheel.brake_torque_2,`
     `._ABS_pre.mtl_front_suspension_upright.brake_torque_1,`
     `._ABS_pre.mtl_front_suspension_upright.brake_torque_1)`

   - `right_front_brake_velocity`
     Funktion:
     `(PI/180)*326*WZ(._ABS_pre.mtr_front_rotor_to_wheel.brake_torque_2,`
     `._ABS_pre.mtr_front_suspension_upright.brake_torque_1,`
     `._ABS_pre.mtr_front_suspension_upright.brake_torque_1)`

   - `left_rear_brake_velocity`
     Funktion:
     `(PI/180)*326*WZ(._ABS_pre.mtl_rear_rotor_to_wheel.brake_torque_2,`
     `._ABS_pre.mtl_rear_suspension_upright.brake_torque_1,`
     `._ABS_pre.mtl_rear_suspension_upright.brake_torque_1)`

   - `right_rear_brake_velocity`
     Funktion:
     `(PI/180)*326*WZ(._ABS_pre.mtr_rear_rotor_to_wheel.brake_torque_2,`
     `._ABS_pre.mtr_rear_suspension_upright.brake_torque_1,`
     `._ABS_pre.mtr_rear_suspension_upright.brake_torque_1)`

3. Fahrzeuggeschwindigkeit an den Mittelpunkten der Räder:
   Es werden die oben beschriebenen *Marker* benutzt, um die Messung der Fahrzeuggeschwindigkeit an den jeweiligen Mittelpunkten der Räder durchzuführen:

   - `F_L_Vel_WC`
     Funktion:
     `-VX._ABS_pre.mtl_front_suspension_upright.mal_Vel_Marker_Front,0,`
     `._ABS_pre.mtl_front_suspension_upright.mal_Vel_Marker_Front)`

   - `F_R_Vel_WC`
     Funktion:
     `-VX(._ABS_pre.mtr_front_suspension_upright.mar_Vel_Marker_Front,0,`
     `._ABS_pre.mtr_front_suspension_upright.mar_Vel_Marker_Front)`

- R_L_Vel_WC
  Funktion:
  -VX(._ABS_pre.mtl_rear_suspension_upright.mal_Vel_Marker_Rear,0,
  ._ABS_pre.mtl_rear_suspension_upright.mal_Vel_Marker_Rear)

- R_R_Vel_WC
  Funktion:
  -VX(._ABS_pre.mtr_rear_suspension_upright.mar_Vel_Marker_Rear,0,
  ._ABS_pre.mtr_rear_suspension_upright.mar_Vel_Marker_Rear)

4. Schlupf:
   Anhand der oben beschriebenen Variablen wird der Schlupf für jedes Rad anhand der folgenden Funktionen berechnet:

   - F_L_D_Lambda
     Funktion:
     IF(VARVAL(._ABS_pre.F_L_Vel_WC):0,0,
     (VARVAL(._ABS_pre.F_L_Vel_WC)-VARVAL(._ABS_pre.left_front_brake_velocity))
     /VARVAL(._ABS_pre.F_L_Vel_WC))

   - F_R_D_Lambda
     Funktion:
     IF(VARVAL(._ABS_pre.F_R_Vel_WC):0,0,
     (VARVAL(._ABS_pre.F_R_Vel_WC)-VARVAL(._ABS_pre.right_front_brake_velocity))
     /VARVAL(._ABS_pre.F_R_Vel_WC))

   - R_L_D_Lambda
     Funktion:
     IF(VARVAL(._ABS_pre.R_L_Vel_WC):0,0,
     (VARVAL(._ABS_pre.R_L_Vel_WC)-VARVAL(._ABS_pre.left_rear_brake_velocity))
     /VARVAL(._ABS_pre.R_L_Vel_WC))

   - R_R_D_Lambda
     Funktion:
     IF(VARVAL(._ABS_pre.R_R_Vel_WC):0,0,
     (VARVAL(._ABS_pre.R_R_Vel_WC)-VARVAL(._ABS_pre.right_rear_brake_velocity))
     /VARVAL(._ABS_pre.R_R_Vel_WC))

Einfügen neuer Arrays

Die Kommunikation zwischen der GSE und dem Modell geschieht über Eingangs- und Ausgangsvektoren. Der Ausgangsvektor vom Modell zum ABS `._ABS_pre.To_ABS` enthält die Schlupfwerte. Der Eingangsvektor `._ABS_pre.From_ABS` enthält die von der DLL berechneten Bremsmomentenwerte für die einzelnen Räder.

Einfügen neuer Aktuatoren für die Bremse

Die Bremsaktorik besteht aus zwei sogenannten *Point-Torques*, deren Eingang aus dem GSE-Ausgangsvektor kommt.

Bremsen-Subsysteme

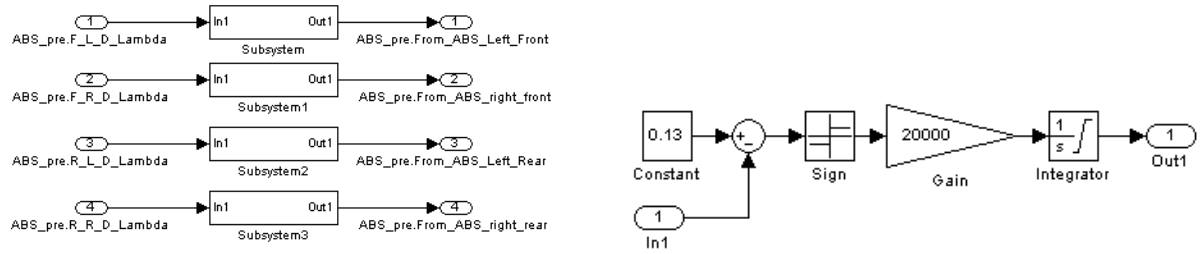
Das Subsystem für die Bremse `TR_Brake_System.sub` wird so verändert, dass es das neue *Template* der Bremse `ABS_pre.tpl` enthält. Das neue Subsystem wird dann in `ABS_pre.sub` umbenannt. Anschließend wird die Fahrzeugdaenbank (*Assembly*) aktualisiert.

Automatisierung der Arbeit

Die oben beschriebenen Änderungen können zur Erleichterung der Arbeit in einem Makro aufgeführt werden.

6.3.3 Anwendung der ZC-HaSh

Implementierung



(a) Gesamtes ABS-Modell in Simulink

(b) ABS-Modell für ein Rad

Bild 6.17: ABS-Modell

Um die in Kapitel 5 vorgestellte ZC-HaSh-Methode zu testen, werden die Diskontinuitäten aus dem ABS-Modell in Simulink (6.17) in Form von ZC-Signale in den Ausgangsvektor der GSE eingespeist. Dazu werden die generierten Codes der *Sign*-Blöcke aus der Abbildung 6.17(b) so verändert, dass die benötigten ZC-Signale als Ausgänge der GSE erscheinen. Für jedes Signal (in unserem Fall vier Signale) wird dann eine entsprechende Behandlung mit den Sensoren in ADAMS durchgeführt. Die Abbildung 6.18 zeigt wie die ZC-HaSh-Methode in ADAMS/Car implementiert wird.

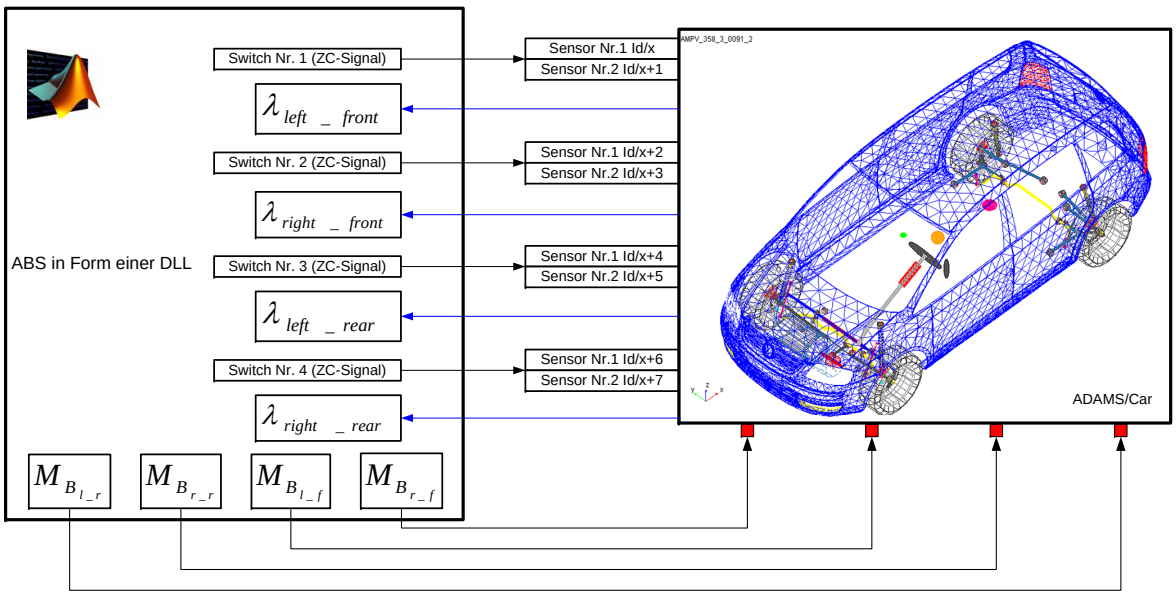


Bild 6.18: Gesamte Struktur der gekoppelten Fahrzeugsimulation

Simulation

Als Fahrmanöver für die Simulation wird das vordefinierte `braking in turn` ausgewählt. Mit den gleichen Einstellungen wird die Simulation bis auf die Schrittweite vier Mal wiederholt:

- Die beiden ersten Simulationen dienen als Referenz für den Vergleich mit den anderen Simulationen. Dazu werden ohne Verwendung der ZC-HaSh Schrittweiten von $0.001s$ und $0.0001s$ eingestellt.
- Die zweite Simulation erzeugt Kurven ohne ZC-HaSh mit einer Schrittweite von $0.1s$.
- Die dritte Simulation erzeugt Kurven unter Verwendung der ZC-HaSh-Technik ebenfalls mit der Schrittweite $0.1s$.

Bei allen Simulationen beträgt das simulierte Zeitintervall $\Delta T = 10s$, der ausgewählte Gleichungslöser ist der `GStiff` und die Integrationsfehlertoleranz wird auf 10^{-2} eingestellt.

6.3.4 Ergebnisse

Die von den Simulationen erzeugten Daten können mit dem Programmpacket *Postprocessor* in ADAMS in Form von Kurven veranschaulicht werden. Die Kurven für die Variablen *Winkelbeschleunigung* und *Winkelgeschwindigkeit* mit den jeweiligen drei Komponenten *Wank*, *Nick* und *Gier* werden in der Abbildung 6.19 dargestellt. Auf die Kurve aus der Simulation mit der Schrittweite $10^{-4}s$ wurde verzichtet, denn der Datensatz war so groß, dass das Laden der Daten problematisch war.

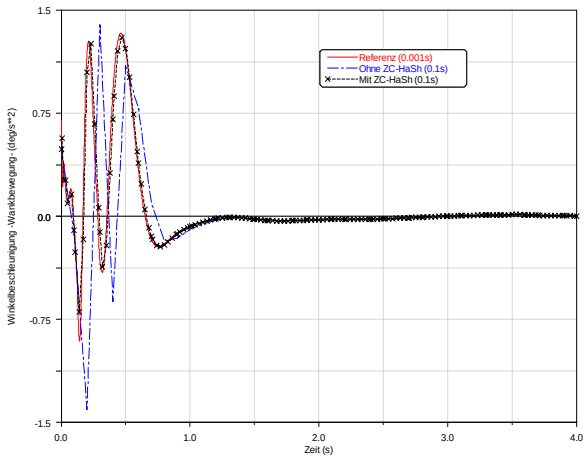
Die Abbildung 6.19 zeigt, dass die Kurven aus der Simulation mit der ZC-HaSh-Methode (Schwarz), verglichen mit den Referenzkurven ($0.001s$) (Rot), einen deutlich besseren Verlauf aufweisen als die Kurven ohne ZC-HaSh (Blau).

Tabelle 6.2: Ergebnisse der gekoppelten Fahrzeugsimulation

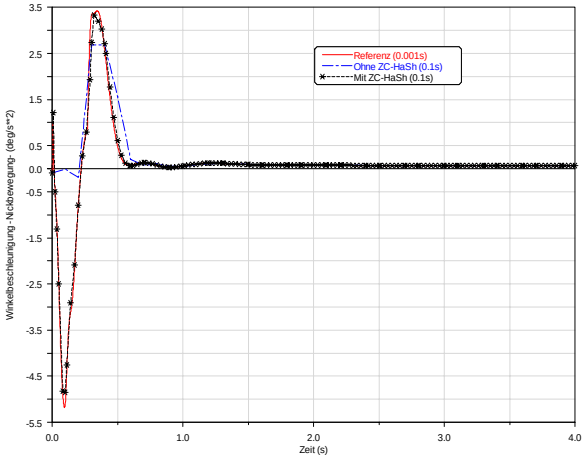
	CPU 1 (P4, 2.8GHz, 1MB RAM)		CPU 2 (AMD)	
Schrittweite	Benötigte Rechenzeit in s	Speicherbedarf in MB	Benötigte Rechenzeit in s	Speicherbedarf in MB
0.1s	27	5	24	3
0.1s mit ZC-HaSh	77	134	49	51
0.001	329	325	617	127
0.0001	3349	3316	-	-

Die Ergebnisse aus der Abbildung 6.19 werden von der Übersichtstabelle 6.2 bestätigt:

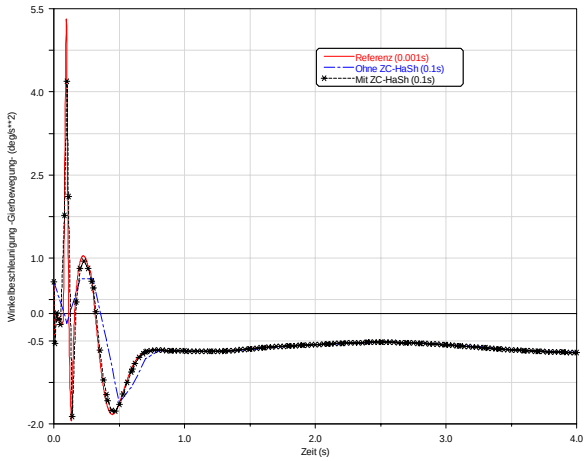
- Die Simulation mit der Schrittweite $0.1s$ ohne die Anwendung der ZC-HaSh-Methode liefert keine plausible Ergebnisse, da die Kurvenverläufe von der Referenz sehr abweichen.
- Die Simulation mit der Schrittweite $0.1s$ unter Verwendung der ZC-HaSh-Methode liefert dagegen gute Ergebnisse. Es werden zwar geringfügig mehr Integrationspunkte



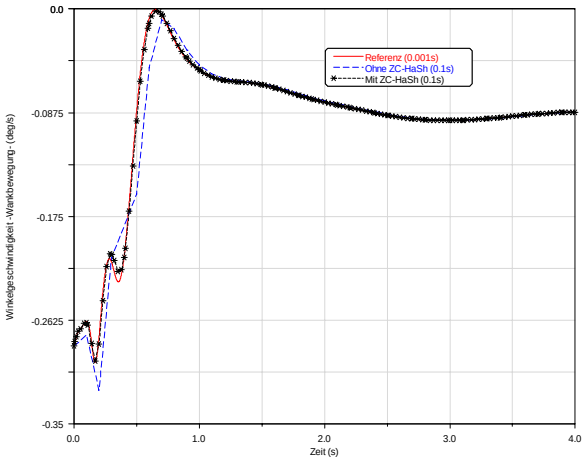
(a) Winkelbeschleunigung (Wank-Komponente)



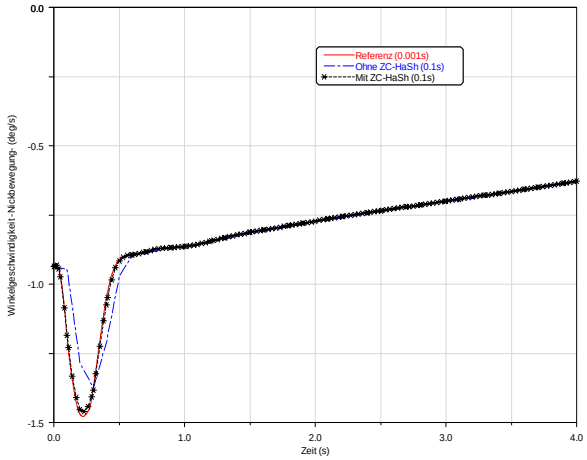
(b) Winkelbeschleunigung (Nick-Komponente)



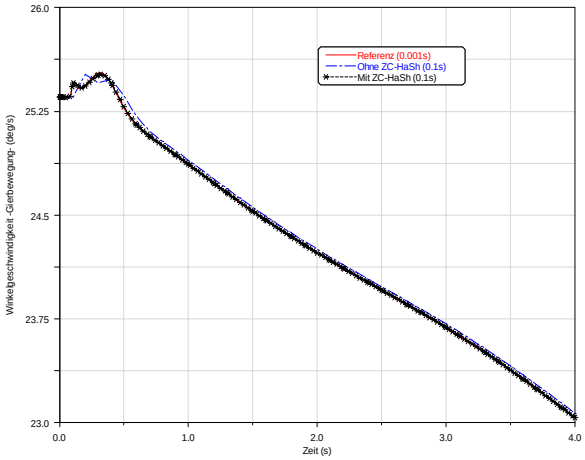
(c) Winkelbeschleunigung (Gier-Komponente)



(d) Winkelgeschwindigkeit (Wank-Komponente)



(e) Winkelgeschwindigkeit (Nick-Komponente)



(f) Winkelgeschwindigkeit (Gier-Komponente)

Bild 6.19: Ergebnisse der Gesamtfahrzeugsimulation

erzeugt und somit mehr Rechenzeit gebraucht, aber der Gewinn an Effizienz der Ergebnisse ist groß.
Verglichen mit der Referenz liefert die Simulation mit der erwähnten Technik einen ähnlichen Kurvenverlauf unter enormer Einsparung der Rechenzeit und des benötigten Speicherbedarfs. So wird in diesem Experiment die Simulation $4,3$ Mal schneller durchgeführt und benötigt $2,42$ Mal weniger Speicher als die Referenzkurve mit der Schrittweite $0.001s$. Der Vergleich mit der Kurve aus der Simulation mit der Schrittweite $10^{-4}s$ liefert noch viel größere Werte. So ist die Simulation mit der ZC-HaSh-Technik $43,5$ Mal schneller und verbraucht auf dem selben Rechner 25 Mal weniger Speicher.

7 Schlussbemerkungen und Ausblick

Sicherlich bieten gekoppelte Simulationen einen großes Potential zur Verbesserung des Entwicklungsprozesses technischer Produkte. Jedoch sind nicht alle Möglichkeiten ausschöpft. Vor allem im automotiven Bereich, wo die Komplexität der Fahrzeugsysteme ständig steigt und die Teilsysteme aus verschiedenen Ingenieurdisziplinen sich immer mehr miteinander verzahnen, ist es wichtig, die Modelle aus den jeweiligen Modellierungsumgebungen zu übernehmen und in einer einheitlichen Umgebung das gesamte System zu simulieren. Um diese Aufgabe zu lösen, muss die Schnittstelle zwischen den verschiedenen *Tools* bestimmte Anforderungen erfüllen, damit die Simulation zu richtigen Ergebnissen führt, ohne zu großen Rechen- und Speicheraufwand zu benötigen.

Der Autor sieht diese Arbeit als einen weiteren Baustein zur methodischen Verbesserung der Schnittstelle zwischen mehreren *Simulationstools*. Die gesamte Fahrzeugsimulation gilt gleichzeitig als Motivation und als Fallstudie für das vorgeschlagene Konzept. Somit ergeben sich folgende Themenschwerpunkte: Mehrkörpersysteme und Regelungssysteme, die miteinander verknüpft werden. Für die Implementierung der Simulationsmodelle wurden die zwei bekannte Programme, die MKS-Software MSC.ADAMS und die regelungstechnische Software MATLAB/Simulink eingesetzt. Das Fundament der Arbeit waren hierbei die Beschreibungsmittel zur Modellierung der Systeme, die Funktionsweise der eingesetzten Gleichungslöser und die Aufbaustruktur der Modelle in beiden *Tools*.
Das präsentierte Konzept *Zero-Crossing-Handshake* beruht darauf, dass stetig differenzierbare Abschnitte in der Simulation mit weiten Integrationsschritten berechnet werden, während in der Nähe von Diskontinuitäten der Gleichungslöser mit kleinen Integrationsschritten fortschreitet. Wichtig dabei ist die Weitergabe der Diskontinuität, die in einem Tool entsteht, an das andere Tool über der Schnittstelle. In diesem Fall werden die Diskontinuitäten, die in dem Simulink-Modell entstehen, über die GSE-Schnittstelle an ADAMS weitergegeben, so dass eine rechtzeitige Verringerung der Schrittweite noch möglich ist. So werden Extrapolationen aus der Vergangenheit nicht mehr nötig und der Solver wird lediglich neugestartet.

Trotz des großen Aufwandes bei der Implementierung liefert die Methode sehr gute Ergebnisse. Unter Bewahrung der Güte der Simulation werden mit dieser Methode große Zeit-

und Recheneinsparungen erzielt. Der Sparfaktor hängt von sehr vielen Faktoren ab: Schärfe und Häufigkeit der Diskontinuitäten, simuliertes Zeitintervall, benutzte Gleichungslöser und weitere Simulationseinstellungen.

Ausgehend von der Tatsache, dass eine Schnittstelle während einer Simulation den Datenaustausch zwischen den beteiligten Softwareprogrammen in einem Toolverbund steuert, ist ihre Standardisierung von großer Bedeutung. Damit werden viele Probleme, wie die Kompatibilität zwischen den Versionen und der spezielle Entwurf der Schnittstelle für jede Softwarekombination vermieden. Die Anforderungen an der Schnittstelle würden bei der Herstellung neuer Software oder neuer Versionen berücksichtigt und das „Andocken“ der Software an die neue standardisierte Schnittstelle würde einfach und reibungslos verlaufen.

Literatur

[1] Y. Bar-Shalom, "On the Track-to-Track Correlation Problem", *IEEE Transactions on Automatic Control*, vol. 26(2), pp. 571-572, 1981.

[2] Edward N. Lorenz, "Deterministic Nonperiodic Flow", *Journal of the Atmospheric Sciences*, Vol. 20, No. 2, 130-141, März 1963.

[3] S. Eberle und P. Göhner, "Softwareentwicklung für eingebettete Systeme mit strukturierten Komponenten", *atp Software-Engineering*, vol. 46(3), pp. 41-52, 2004.

[4] J. Wedekind, "Konservative Synchronisation und Time Warp", *Seminar Verteilte Simulation*, pp. 1-17, 1999.

[5] A. Wohnhaas und R. Moser, "Modellaustausch zwischen Steuergeräte-Entwicklungstools auf Basis einheitlicher grafischer Blockbibliotheken".

[6] J. Axelsson, "Methods and Tools for Systems Engineering of Automotive Electronic Architectures", *Design Automation and Test in Europe Conference*, pp. 13-16, 2001.

[7] J. Jung, S. Yoo und K. Choi, "Performance Improvement of Multi-Processor Systems Cosimulation based on SW Analysis", *Design, Automation and Test in Europe, 2001. Conference and Exhibition*, pp. 749-753, 2001.

[8] A. Hoffman, T. Kogel und H. Meyr, "A framework for fast hardware-software cosimulation", *IEEE Transactions on Automatic Control*, pp. 760-765, 2001.

[9] J. Bortolazzi, T. Hirth und T. Raith, "Specification and design of electronic control units", *Proceedings of the conference on European design automation*, pp. 36-41, 1996.

[10] K. Grimm, "Software Technology in an Automotive Company - Major Challenges", *25th International Conference on Software Engineering (ICSE'03)*, pp. 498-505, 2003.

[11] A. Panday, D. Couderc und S. Marichalar, "AIL: description of a global electronic architecture at the vehicle scale", *DBLP, http://dblp.uni-trier.de*, pp. 112, 2001.

[12] C. Clauß, S.Reitz und P. Schwarz, "Simulation mechanisch-elektrischer Wechselwirkungen am Beispiel eines sensorischen Mikrosystems", *SIM2000-Simulation im Maschinenbau*, pp. 183, 2000.

[13] S. Liebig, S. Dronka, S. Helduser und M. Stüwing, "Simulation gekoppelter mechanischer und hydraulischer Systeme", *Grundlagen und Methoden der Modellierung und Simulation*, 2001.

[14] S. Dierssen, "Die virtuelle Maschine: Konfiguration, Simulation, Visualisierung", *SimVis 2000*, pp. 145-156, 2000.

[15] Nimrod Lilith, Jonathan Billington, Jörn Freiheit, "Approximate Closed-Form Aggregation of a Fork-Join Structure in Generalised Stochastic Petri Nets", *ACM International Conference Proceeding Series*, Vol. 180, Article No. 32, 2006.

[16] Dressler P., "The Entity-Relationchip Approach System Analysis and Design", *Proc. 1st International Conference on the Entity-Relationship Approach*, North-Holland, 1980.

[17] Sreenivas R. S. und Krogh B. H., "Discrete Event Dynamic Systems: Theory and Application", *On condition/event systems with discrete state realizations*, pp 209-236, 1991.

[18] Arenz A. und Schnieder E., "Integrating different single purpose engineering tools: A case study for the computer aided system design of a 'goliath' robot", *The 2nd Tampere International Conference on Machine Automation (ICMA ´98)*, pp 609-623, 1998.

[19] H. E. Scherf, *Modellbildung und Simulation dynamischer Systeme*, Oldenbourg Verlag, 2003.

[20] Verein Deutscher Ingenieure: VDI Richtlinie, *Simulation von Systemen in Logistik, Materialfluss und Produktion*, Düsseldorf, VDI-Verlag, 1993. ISBN 3-18-091087-9

[21] G. Kempf, *Simulation von Prototypen vernetzter Kfz-Steuergeräte*, Technische Universität München, 1996.

[22] A. B. Arenz, *Simulation und Regelung starr-elastischer Mehrkörpersysteme mit integrierten Werkzeugen*, Papierflieger Verlag GmbH, 2001.

[23] M. Chouika, *Entwurf diskret-kontinuierlicher Steuerungssysteme- Modellbildung, Analyse und Synthese mit hybriden Petrinetzen*, Fortschritt-Berichte VDI, Reihe 8: Meß-, Steuerungs- und Regelungstechnik, Nr.797, 1999.

[24] Model-based and System-Based Design, *Using Simulink (version 5)*, The MathWorks, 2002.

[25] Model-based and System-Based Design, *Real-Time Workshop User's Guide*, The MathWorks, 2002.

[26] D. Negrut und A. Dyer, *ADAMS/Solver Primer*, MSC-Software, 2004.

[27] G. Bikker und M. Schröder, *Methodische Anforderungsanalyse und automatisierter Entwurf sicherheitsrelevanter Eisenbahnleitsysteme mit kooperierenden Werkzeugen*, VDI Verlag, 2002.

[28] E. Brommundt und G. Sachs, *Technische Mechanik*, Springer Verlag, 1991.

[29] Dean C. Karnopp, Donald L. Margolis and Ronald C. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems*, fourth edition, John Wiley and sons, 2006.

[30] Y. Du, *Zur Optimierung des dynamischen Verhaltens von Gesamtfahrzeugen mit mechatronischen Komponenten*, Shaker-Verlag, 2006.

[31] http://de.wikipedia.org/wiki/Schmetterlingseffekt.

[32] http://de.wikipedia.org/wiki/Mechatronik.

[33] http://en.wikipedia.org/wiki/Mechatronics.

[34] G. H. Golub, J. M. Ortega, *Wissenschaftliches Rechnen und Differentialgleichungen. Eine Einführung in die Numerische Mathematik*, Heldermann Verlag, Lemgo 1995.

[35] E.L. Ince, *Ordinary Differential Equations*, Dover Publications, 1956.

[36] A. Papoulis, *The Fourier Integral and Its Applications*, McGraw-Hill, New York, 1962.

[37] F. Erwe, *Gewöhnliche Differentialgleichungen*, Bibliographisches Institut, Mannheim, 1964.

[38] S. Bochner, K.Chandrasekharan, *Fourier Transforms*, Princeton Book Comp. Publication, 2001.

[39] B. Bauke, *Die Mathematik des Naturforschers und Ingenieurs (Band VI): Partielle Differentialgleichungen*, S. Hirzel Verlag Leipzig, 1965.

[40] G. Bräuning, *Gewöhnliche Differentialgleichungen*, Verlag Harry Deutsch, Frankfurt und Zürich, 1964.

[41] G. Doetsch, *Handbuch der Laplace-Transformation*, Verlag Bikhäuser, Basel, 1950.

[42] O. Föllinger, *Laplace- und Fourier-Transformation*, Verlag Elitera, Berlin, 1977.

[43] http://de.wikipedia.org/wiki/Laplace-Transformation.

[44] Karnopp, D. C., and Rosenberg, R. C., *Analysis and Simulation of Multiport Systems: The Bond Graph Approach to Physical System Dynamics*, the M.I.T. Press, Cambridge, 1968.

[45] K. Küpfmüller, *Die Systemtheorie der elektrischen Nachrichtenübertragung*, Hirzel-Verlag, Stuttgart, mehrere Auflagen ab 1949.

[46] G. H. Mealy, *A Method for Synthesizing Sequential Circuits*, Bell System Tech., pp. 1045 bis 1079, September 1955.

[47] Claude E. Shannon, John McCarthy, *Studien zur Theorie der Automaten*, Rogner und Bernhard, München 1974. (Übersetzung einer 1956 im Original erschienenen Anthologie mit Beträgen u.a. von J. v. Neumann, S.C. Kleene, E.F. Moore, M. Minsky)

[48] Scheuring, R., *Modellierung, Beobachtung und Steuerung ereignisorientierter verfahrenstechnischer Systeme.* Fortschritt-Berichte VDI, Reihe 8, Nr. 475, VDI Verlag, Düsseldorf, 1995.

[49] Petri, Carl Adam, *Kommunikation mit Automaten*, Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn, 1962.

[50] Bochmann D., *Der Boolesche Differentialkalkül - Ein Überblick*, Oldenbourg Verlag, München, Wien, 1998.

[51] Dressler H., *Problemlösen mit Entscheidungstabellen*, Oldenbourg Verlag, München, 1973.

[52] Conner M. F., *Structured Analysis and Design Technique*, Waltham, 1980.

[53] Krüger F., *Temporal Logic of Programs*, Springer Verlag, Berlin, Heidelberg 1987.

[54] Marsan A. M., Balbo und G. Conte *Performance Models of Multiprocessor Systems*, The Mit Press Series In Computer Systems, Cambridge, 1986.

[55] Sutcliffe A., *Jackson System Development*, Prentice Hall, New York, 1988.

[56] Raasch J., *Systementwicklung mit strukturierten Methoden*, Carl Hanser Verlag, Wien, 1991.

[57] Abrial J.-R., *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[58] Harry A., *Formal methods fact file-VDM and Z.*, John Wiley and Sons, 1996.

[59] Lightfoot D., *Formal Specification Using Z*, Macmillan Distribution Ltd, 1991.

[60] Van Eijk P. H. J. C.A: Vissers und M. Diaz, *The formal formal description technique LOTOS*, Elsevier Science Publishers B.V., 1989.

[61] Fishwick P. A., *A simulation environment for multimodeling* in Discrete Event Dynamic System: theory and applications, vol. 3, pp. 151 bis 171, 1993.

[62] Burkhardt R., *UML-Unified Modeling Language.*, Addison-Wesley, Bonn, 1998.

[63] Verband Deutscher Ingenieure.

[64] Barth T. und Grauer M., *Grid Computing-Ansätze für verteiltes virtuelles Prototyping*, Universität Siegen, Springer Verlag, August 2002.

[65] Dieter Bestle, *Analyse und Optimierung von Mehrkörpersysteme*, Springer-Verlag, 1994.

[66] Martin Förg, *Mehrkörpersimulation*, Vorlesungsskript, Lehrstuhl für angewandte Mechanik, technische Universität München, 2006.

[67] MSC.ADAMS, *help*, version 2005r2, 2005.

[68] Wikimedia Foundation Inc., *http://de.wikipedia.org/* , 2006.

[69] Friedrich Baumjohann, *Einführung in DADS*, Vorlesungsskript, 2006.

[70] W. Schumacher, *Grundlagen der Regelungstechnik*, Vorlesungsskript, 1999.

[71] W. Schumacher, *Regelungstechnik 1*, Vorlesungsskript, 2002.

[72] Kai Müller, *Entwurf Robuster Regelungen*, B.G. Teubner, Stuttgart 1996.